

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Toshiaki IIZUKA

Group Art Unit: 2122

Serial No.: 10/815,506

Examiner: TBD

Filed: March 31, 2004

Confirmation No. 1302

LOG ACQUISITION METHOD AND ITS CONTROL  
PROGRAM AND STORAGE MEDIUMCERTIFICATE OF MAILING (37 C.F.R. §1.8(A))

Commissioner for Patents  
 P.O. Box 1450  
 Alexandria, VA 22313-1450

Sir:

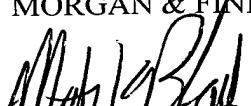
I hereby certify that the attached:

1. Claim to Convention Priority
2. Certified Priority document - Japanese Patent Application  
 Serial No. 2003-099465, filed April 2, 2003
3. Change of Correspondence Address
4. Return receipt postcard

along with any paper(s) referred to as being attached or enclosed and this Certificate of Mailing are being deposited with the United States Postal Service on date shown below with sufficient postage as first-class mail in an envelope addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Respectfully submitted,  
 MORGAN & FINNEGAN, L.L.P.

By:



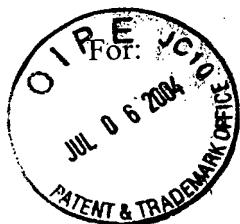
Matthew K. Blackburn  
 Reg. No. 47,428

Dated: June 2 2004Correspondence Address:

MORGAN & FINNEGAN, L.L.P.  
 345 Park Avenue  
 New York, NY 10154-0053  
 (212) 758-4800 Telephone  
 (212) 751-6849 Facsimile

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Toshiaki IIZUKA	Group Art Unit	2122
Serial No.:	10/815,506	Examiner:	TBD
Filed:	March 31, 2004	Confirmation No.	1302



LOG ACQUISITION METHOD  
AND ITS CONTROL PROGRAM AND STORAGE MEDIUM

CLAIM TO CONVENTION PRIORITY

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

In the matter of the above-identified application and under the provisions of 35 U.S.C. §119 and 37 C.F.R. §1.55, applicant(s) claim(s) the benefit of the following prior application(s):

Application(s) filed in:	Japan
In the name of:	Canon Kabushiki Kaisha
Serial No(s):	2003-099465
Filing Date(s):	April 2, 2003

Pursuant to the Claim to Priority, applicant(s) submit(s) a duly certified copy of said foreign application.

A duly certified copy of said foreign application is in the file of application Serial No. \_\_\_\_\_, filed \_\_\_\_\_.

Respectfully submitted,  
MORGAN & FINNEGAN, L.L.P.

Dated: July 2 2004

By:

  
Matthew K. Blackburn  
Registration No. 47,428

Correspondence Address:

MORGAN & FINNEGAN, L.L.P.  
345 Park Avenue  
New York, NY 10154-0053  
(212) 758-4800 Telephone  
(212) 751-6849 Facsimile

日本国特許庁  
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日 2003年 4月 2日  
Date of Application:

出願番号 特願 2003-099465  
Application Number:

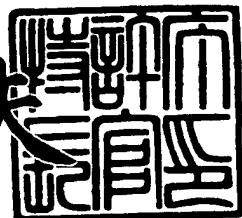
[ST. 10/C] : [JP 2003-099465]

出願人 キヤノン株式会社  
Applicant(s):

2004年 4月 19日

特許庁長官  
Commissioner,  
Japan Patent Office

今井 康夫



【書類名】 特許願  
【整理番号】 253789  
【提出日】 平成15年 4月 2日  
【あて先】 特許庁長官殿  
【国際特許分類】 G06F 9/00  
【発明の名称】 ログ取得方法  
【請求項の数】 1  
【発明者】  
【住所又は居所】 東京都大田区下丸子3丁目30番2号 キヤノン株式会社内  
【氏名】 飯塚 利明  
【特許出願人】  
【識別番号】 000001007  
【氏名又は名称】 キヤノン株式会社  
【代理人】  
【識別番号】 100076428  
【弁理士】  
【氏名又は名称】 大塚 康徳  
【電話番号】 03-5276-3241  
【選任した代理人】  
【識別番号】 100112508  
【弁理士】  
【氏名又は名称】 高柳 司郎  
【電話番号】 03-5276-3241  
【選任した代理人】  
【識別番号】 100115071  
【弁理士】  
【氏名又は名称】 大塚 康弘  
【電話番号】 03-5276-3241

## 【選任した代理人】

【識別番号】 100116894

## 【弁理士】

【氏名又は名称】 木村 秀二

【電話番号】 03-5276-3241

## 【手数料の表示】

【予納台帳番号】 003458

【納付金額】 21,000円

## 【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 0102485

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 ログ取得方法

【特許請求の範囲】

【請求項 1】 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内の関数のうち、指定された関数を識別する工程と、

ロードされた前記所定の処理を行う関数のアドレスと前記指定されたオペレーティングシステム内の関数のアドレスとを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程とを備えることを特徴とするログ取得方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、複数にモジュール分けされたソフトウェアの処理ログを取得するための技術に関するものである。

【0002】

【従来の技術】

従来より、再現率の低いソフトウェアの障害に対しては、ソフトウェアの処理ログを取得し、当該処理ログを解析することによって、障害の原因をつきとめ、その対策を講じてきた（例えば、特許文献1を参照）。

【0003】

**【特許文献1】**

特開平11-296415号公報

**【0004】****【発明が解決しようとする課題】**

しかし、上記従来の処理ログの取得には以下のような問題点がある。

(1) ユーザの動作環境でも処理ログを取得しつづけるためには、ソフトウェアのモジュール自体に手を加え、処理ログ取得ルーチンを追加しなければならず、処理ログ取得のための作業負荷が大きかった。

(2) 処理ログ取得はモジュール毎に行うため、生成された処理ログはモジュール単位のものとなってしまい、ソフトウェア全体の処理を、完全に時間順の処理ログとして取得するのが困難である。このため、全体の処理ログとしての見通しが悪く、処理ログを解析して障害の原因を発見するまでのプロセスに工数がかかっていた。

**【0005】**

本発明は、上記課題を鑑みてなされたものであり、複数にモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能なログ取得方法、ならびに該方法をコンピュータによって実現させるためのプログラム、および該プログラムを格納した記憶媒体を提供することを目的とする。

**【0006】****【課題を解決するための手段】**

上記の目的を達成するために本発明に係るログ取得方法は以下のよう構成を備える。即ち、

所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内の関数のうち、指定された関数を識別する工程と、

ロードされた前記所定の処理を行う関数のアドレスと前記指定されたオペレーティングシステム内の関数のアドレスとを、ログ取得のための関数のアドレスに

書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程とを備える。

### 【0007】

#### 【発明の実施の形態】

##### 【第1の実施形態】

本実施形態は、あるモジュールから別のモジュール内に存在する関数の呼び出しが行われる際の仕組みである、メモリに保持されたインポート関数アドレス、または仮想関数アドレステーブル（Virtual Address Table）を利用して、モジュール間の関数呼び出しをフックして処理ログに記録することで、ソフトウェアのモジュール自体に手を加えることなく、ソフトウェア全体の処理を、時間順の処理ログとして取得することを可能にするものである。以下に具体的に説明する。

### 【0008】

#### ＜システム構成＞

図1は、本発明の各実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。説明を簡略化するために、本実施形態では、本ソフトウェア評価システムが1台のPC内部に構築されるものとするが、本発明にかかるログ取得方法は1台のPC内部に構築されるか、あるいは複数のPCにネットワークシステムとして構築されるかによらず有効である。

### 【0009】

本ログ取得方法を実現するソフトウェア評価システムには、CPU1、チップセット2、RAM3、ハードディスクコントローラ4、ディスプレイコントロー

ラ5、ハードディスクドライブ6、CD-ROMドライブ7、ディスプレイ8が搭載されている。また、CPU1とチップセット2とを繋ぐ信号線11、チップセット2とRAM3とを繋ぐ信号線12、チップセット2と各種周辺機器とを繋ぐ周辺機器バス13、ハードディスクコントローラ4とハードディスクドライブ6とを繋ぐ信号線14、ハードディスクコントローラ4とCD-ROMドライブ7とを繋ぐ信号線15、ディスプレイコントローラ5とディスプレイ8とを繋ぐ信号線16が搭載されている。

#### 【0010】

##### ＜関数処理に対する処理ログ取得＞

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理を説明するために、まず図2によって、複数のモジュールに分かれたソフトウェアが、通常の状態でどのようにメモリにロードされるかを説明する。

#### 【0011】

通常、複数のモジュールに分かれたソフトウェアは、全体の制御を行う実行ファイルEXE(23)と、モジュールとして存在しEXEの補完的な役割を担うダイナミックリンクライブラリDLL(27)とに分かれており、メモリにはEXEとDLLの両方がロードされる。EXEはコードセグメント(28)とデータセグメント(29)、そしてインポート関数アドレステーブル(22)とから成っている。更に、インポート関数アドレステーブル(22)は、関数の所属するDLLによって分かれており(21, 24)、DLLごとにそれぞれの関数がロードされたアドレスが書かれている(30~35)。DLLの関数の実体は、DLLごとに分かれて(25, 26)ロードされ、それぞれの関数は該当するDLLの一部としてロードされる(36~41)。この図では、1本のEXEがA. DLL及びB. DLLの2つのダイナミックリンクライブラリ内の関数を使用している例を示しており、実際に使用される関数はFunc\_AA, Func\_AB, Func\_AC, Func\_BA, Func\_BB, Func\_BCの6個となっている。

#### 【0012】

EXEのコードセグメント内にあるコードが関数Func\_AAを呼び出す場合には、まずインポート関数アドレステーブル(22)内に書かれたFunc\_AAのアドレス(30)が読み込まれる。ここには実際にはA.DLLの一部として読み込まれたFunc\_AAコード(36)のアドレスが書かれており、そのアドレスをコールすることによって、EXEのコードはA.DLLのFunc\_AAを呼び出すことができる。

#### 【0013】

図3は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図2とは、ログ取得用のコードに対してIAT Patch(Import Address Table Patch)という手法を用いて、関数呼び出しをリダイレクトしているという点で異なっている。

#### 【0014】

ログ取得が開始されると、メモリ内にはIAT Patch用のDLLであるC.DLL(58)がロードされる。C.DLL(58)はインポート関数アドレステーブル(52)内に書かれた関数のアドレスを、C.DLL(58)内のログ取得コードであるFunc\_CAA, Func\_CAB, Func\_CAC, Func\_CBA, Func\_CBB, Func\_CBCのアドレスに書き換える(61～66)。C.DLL(58)内のFunc\_CAA, Func\_CAB, Func\_CAC, Func\_CBA, Func\_CBB, Func\_CBCのコード(73～78)は、ログを記録すると共に、元々の関数呼び出しを受けるべくメモリにロードされている、該当する関数であるFunc\_AA, Func\_AB, Func\_AC, Func\_BA, Func\_BB, Func\_BC(67～72)を呼び出す。

#### 【0015】

図4Aは、図3におけるIAT Patchの処理をあらわす図であり、図4Bはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではEXEがA.DLL(55)内のFunc\_AAを呼び出す際に、IAT Patchによるログ取得コードがどのように動作するかの例をあらわ

している。

### 【0016】

EXE（図4Aの91）がFunc\_AAをコールすると（図4Aの94）、C. DLL（58）内にあるログ取得コードがDLL名（C. DLL）／関数名（Func\_AA）をメモリに保存し（図4BのステップS402）、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する（図4Aの95、図4BのステップS403）。その後C. DLLは本来呼び出されるはずであった、A. DLL（図4Aの93）内のFunc\_AAをコールする（図4Aの96、図4BのステップS404）。A. DLLのFunc\_AA処理（図4Aの97）が終了し、C. DLLに制御がリターンすると（図4Aの98）、C. DLLはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する（図4Aの99）。その後、C. DLLは保存したログ情報をファイルに書き込み（図4Aの100、図4BのステップS405）、あたかもA. DLLのFunc\_AAが通常通りに終了したかのように、EXEにリターンする（101）。

### 【0017】

図5は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの機能構成をあらわす図である。通常は実行形式のEXE（113）が、DLL-1（116）やDLL-2（117）内の関数を呼び出すが、ここではAPIトレーサと呼ばれるログ取得コードを埋め込み（114）、処理ログを生成している（115）。APIトレーサは、DLL-1やDLL-2の関数定義を記述したファイル（111）と、どのDLLのどの関数のインポート関数テーブルを書き換えてログを取得するかの設定シナリオ（トレースシナリオ112）を元に動作する。

### 【0018】

#### ＜メソッド処理に対する処理ログ取得＞

次に、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、実行ファイルEXEがCOM（Component O

object Model) サーバでエクスポートされているインターフェースのインスタンス作成時に、どのようにメモリにロードされるかを説明するために、まず、図6によって、通常の状態でどのようにメモリにロードされるかを説明する。

### 【0019】

通常、インターフェースのインスタンス作成を行うと、COMサーバ内で、要求されたインターフェース(121, 122)と、そのメソッド(:オブジェクト指向プログラミングにおいて、オブジェクトの実行する手続きを記述したプログラム、130~135)が作成され、それらは、メモリ上に両方がロードされる。ここで、virtual address table(仮想アドレステーブル)は作成された各インターフェース毎に作られ(118, 120)、作成要求を行ったEXEに渡される。このvirtual address tableには各メソッドの作成されたアドレスが書かれている(124~129)。EXEはこれら情報を利用し、各インターフェースに対して呼び出しを行う。この図では、1本のEXEがInterface A及びInterface Bの2つのインターフェースのインスタンスを作成しており、そのインターフェース内部のメソッドを使用している例を示しており、実際に使用されているメソッドは、Method AA, Method AB, Method AC, Method BA, Method BB, Method BCとなっている。

### 【0020】

EXEのコードが関数Method AAを呼び出す場合には、まずvirtual address table内に書かれたMethod AAのアドレス(124)が読み込まれる。ここには実際にはCOMサーバのInterface Aの一部として作成されたMethod AAコード(130)のアドレスが書かれており、そのアドレスをコールすることによって、EXEのコードはInterface AのMethod AAを呼び出すことができる。

### 【0021】

図7は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのメモリ構成をあらわす図であり、図6とは、ログ取得用のコー

ドに対してVTable Patch (virtual address table Patch) という手法を用いて、メソッド呼び出しをリダイレクトしているという点で異なっている。

#### 【0022】

ログ取得が開始されると、メモリ内にはVTable Patch用のDLL (143) がロードされる。このDLLはvirtual address table (136, 138) 内に書かれたメソッドのアドレスを、DLL内のログ取得コードであるMethod A'A, Method A'B, Method A'C, Method B'A, Method B'B, Method B'Cのアドレスに書き換える (145～150)。DLL内のMethod A'A, Method A'B, Method A'C, Method B'A, Method B'B, Method B'Cのコード (157～162) は、ログを記録すると共に、元々のメソッド呼び出しを受けるべくメモリにロードされている、該当するメソッドであるMethod AA, Method AB, Method AC, Method BA, Method BB, Method BC (157～162) を呼び出す。

#### 【0023】

図8Aは、図7におけるVTable Patchの処理をあらわす図、図8Bはログ取得処理の流れを示すフローチャートである。説明を簡略化するために、この図ではEXEがCOMサーバ内のInterface AのMethod AAを呼び出す際に、VTable Patchによるログ取得コードがどのように動作するかの例をあらわしている。

#### 【0024】

EXE (図8Aの163) がMethod AAをコールすると (図8Aの166)、DLL内にあるログ取得コードがモジュール名/インターフェース名/メソッド名をメモリに保存し (図8BのステップS802)、呼び出し時刻をメモリに保存し、呼び出し時のパラメータをメモリに保存し、呼び出し時のポインタパラメータの指すメモリ内容を、別メモリに保存する (図8Aの167、図8BのステップS803)。その後DLLは本来呼び出されるはずであった、CO

Mサーバ（図8Aの165）内のMethod AAをコールする（図8Aの168、図8BのステップS804）。COMサーバのMethod AA処理（図8Aの169）が終了し、DLLに制御がリターンすると（図8Aの170）、DLLはリターン時の時刻をメモリに保存し、戻り値をメモリに保存し、リターン時にポインタパラメータが指すメモリ内容を、別メモリに保存する（図8Aの171）。その後、DLLは保存したログ情報をファイルに書き込み（図8Aの172、図8BのステップS805）、あたかもCOMサーバのMethod AAが通常通りに終了したかのように、EXEにリターンする（図8Aの173）。

#### 【0025】

図9は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの機能構成をあらわす図である。通常は実行形式のEXE（176）が、COMサーバ1（179）やCOMサーバ2（180）内のメソッドを呼び出すが、ここではAPIトレーサと呼ばれるログ取得コードを埋め込み（177）、処理ログを生成している（178）。APIトレーサは、COMサーバ1（179）やCOMサーバ2の関数定義を記述したファイル（174）と、どのCOMサーバのどのインターフェースのどのメソッドのvirtual address tableを書き換えてログを取得するかの設定シナリオ（175）を元に動作する。

#### 【0026】

##### ＜実施例＞

図10は、本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムに対して、それぞれの関数及びメソッドのパラメータの形式や、戻り値の形式を指示する、関数定義ファイルの例を示す図である。DLL/インターフェース名及び関数/メソッド名を記述し（「関数/メソッド」とは、「関数またはメソッド」の意、以下同じ）、その関数/メソッドに対する、パラメータ及び戻り値の型が示されている。本実施形態にかかるログ取得方法を実現するソフトウェア評価システムは、この関数定義ファイルによって指示された内容を元に、それぞれの関数/メソッドがどのようなパラメータ/戻り値を有してい

るかを判断し、その内容をログとして取得する。

### 【0027】

図11は、図10に示した関数定義ファイルを用いて、本発明の実施形態にかかるソフトウェア評価システムで取得した、ログの一例を示す図である。それぞれの呼び出しに対して、関数／メソッドが呼び出された時刻（In時刻、Out時刻）、及びその際のパラメータ／戻り値が、ログとして生成される。

### 【0028】

以上の説明から明らかなように、本実施形態にかかるログ取得方法によれば、複数にモジュール分けされたソフトウェアの処理ログの取得において、モジュール自体に変更を加えることなく、モジュール内に用意された関数／メソッドの呼び出しを処理ログとして記録することが可能となり、処理ログ取得のための作業負荷を低減することが可能となる。また、生成される処理ログは、時間順の処理ログとして取得することができ、処理ログの解析が容易となるため、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

### 【0029】

#### 【第2の実施形態（その1）】

上記第1の実施形態では、EXEによって呼び出される関数／メソッドすべてについて処理ログを取得することとした。しかしながら、この場合、取得される処理ログの数が膨大になってしまうことも考えられる。そこで、本実施形態ではEXEによって呼び出される関数／メソッドのうち、処理ログの取得対象を限定する場合について述べる。

### 【0030】

一般にソフトウェアのモジュール構成は、OS SurfaceによってOS部分のDLLとアプリケーション部分のDLLとに分けられる。図12は、一般的なソフトウェアのモジュール構成の一例を示す図である。同図において、1200はAPP. EXEであり、1201～1203はそれぞれ、Module A. d11、Module B. d11、Module C. d11であり、アプリケーション部分のDLLである。一方、1205～1210はそれぞれ、User32. d11、GDI32. d11、Nt d11. d11、Ws2\_32. d1

1、Unidrv.dllであり、OS部分のDLLである。

#### 【0031】

そこで、本実施形態では、このようなモジュール構成を備えるソフトウェアにおいて、アプリケーション部分のDLL内の関数が呼び出された場合のみ処理ログを取得することで処理ログの収集対象を限定する。

#### 【0032】

図13は、オペレーティング・システムのモジュールを定義する例を示す図である。このモジュール定義をもって、図14に示した処理が行なわれる。

#### 【0033】

図14は、本発明の第2の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

#### 【0034】

ステップS1400で処理が開始されると、まず、図13で示したオペレーティング・システムのモジュール定義1300を取得し、OSモジュールリストを保持する（ステップS1401）。続いて、関数／メソッドの呼び出し元のモジュールが、そのOSモジュールリストに存在するか否かを判別し（ステップS1402、S1403）、存在する場合は呼び出し先のモジュールをOSモジュールリストに追加する（ステップS1409）だけで、そのまま元の関数／メソッドを呼び出す（ステップS1410）。このステップS1409の処理により、更にそのモジュール以下の呼び出しが行われた場合でも、その呼び出しの処理ログを取得しない手段が確立されている。

#### 【0035】

一方、ステップS1403においてOSモジュールリストに存在しない場合は、DLL名／インターフェース名／関数名／メソッド名／呼び出し時の時刻をHDDに保存し（ステップS1404）、その呼び出しに対してのパラメータをHDDに保存する（ステップS1405）。

#### 【0036】

続いて、本体の関数／メソッドを呼び出す（ステップS1406）。関数／メソッド内部の処理が終了すると、DLL名／インターフェース名／関数名／メソ

ツド名／終了時の時刻をHDDに保存し（ステップS1407）、その呼び出しに対してのパラメータ及び戻り値をHDDに保存する（ステップS1408）。続いて、関数／メソッドのリターン処理を行なう（ステップS1411）。この処理は、評価対象となるプログラムが終了するまで続行される（ステップS1412）。

#### 【0037】

図15Aは、図14における処理によって取得された処理ログの例である。なお、比較のために図15BにOS部分のDLL内の関数が呼び出された際にも処理ログを取得することとした場合の例を示す。図15Aと図15Bとを対比することで明らかなように、図13におけるオペレーティング・システムのモジュール定義1300により、図15AではKernel32.dllからNtdll.dllに対する呼び出し、あるいはUser32.dllからWs2\_32.dllに対する呼び出しといった、オペレーティング・システムの内部の処理ログも含めた処理ログが取得されていないため、よりサイズの小さい処理ログで解析することが可能となっている。

#### 【0038】

以上の説明から明らかなように、本実施形態によれば、関数定義ファイルにおいて「オペレーティング・システムのサーフェイス」を指定することで、ソフトウェアに対する適切な処理ログが取得可能となり、バグ原因特定などの解析が可能となる。

#### 【0039】

##### 【第2の実施形態（その2）】

上記第2の実施形態（その1）においては、オペレーティング・システムのサーフェイス以下すべての処理ログを取得しない例を挙げたが、実際はサーフェイス以下の一部（OS部分のDLL）に関しても、処理ログを取得したい場合がある。

#### 【0040】

例えば、上記第1の実施形態において示したログ取得方法を実現するためのプログラム（以下、「ログ取得プログラム」と称す）は、ドライバSDK（SDK

= Software Development Kit) に組み込んで利用する場合ことが考えられる。

#### 【0041】

一般にドライバSDK (例えばプリンタSDK) は周辺機器メーカー (プリンタメーカー) により提供されるが、ドライバSDKを用いるソフトウェア (例えば、プリンタに印刷する文書や画像を生成、編集するためのソフトウェア) は他のメーカーが作成することが多い。このため、ソフトウェアがドライバSDK (プリンタSDK) を使用した際に正常に動作しなかった時は、その原因がドライバSDK (プリンタSDK) 側のバグによるものなのか、ソフトウェア側のバグによるものなのかを判別することが重要となってくる。このようなケースにおいてはログ取得プログラムにより、プリンタSDKの処理内容を処理ログとして取得しておくことが重要となってくる。そのためには、Unidrv. dll (1210) などのOS部分のDLL内の関数を呼び出した場合の処理ログも収集対象としておく必要がある。

#### 【0042】

つまり、処理ログの取得対象を限定するにあたっては、ログ取得プログラムの使用目的に応じて、任意に選択できるようにしておくことが望ましい。本実施形態では、かかる点に考慮したログ取得プログラムについて説明する。

#### 【0043】

図16は、このような必要性に鑑みて、オペレーティング・システムの除外モジュール (ログ取得対象外として定義されるオペレーティング・システムのモジュールから除外するモジュール、つまり、ログ取得対象となるモジュール) を定義する例を示す図である。この除外モジュール定義及び図13に示したサーフェイス定義をもって、図17にあらわした処理が行なわれる。本実施形態では、サーフェイスから除外の指定をユーザが行なうことにより、上記のケースにおいてドライバの処理ログをソフトウェアの処理ログと併せて取得する。

#### 【0044】

図17は、本発明の第2の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

**【0045】**

ステップS1700において処理が開始されると、まず本ログ取得プログラムは、図13で示したオペレーティング・システムのモジュール定義1300を取得し、OSモジュールリストを保持する（ステップS1701）。続いて、図16で示した除外モジュール定義を取得し、除外モジュールリストを保持する（ステップS1702）。続いて、関数／メソッドの呼び出し元のモジュールが存在するかどうかを、OSモジュールリスト及び除外モジュールリストの両リストから検索する（ステップS1703）。

**【0046】**

続いて、呼び出し元のモジュールがOSモジュールリストに存在するか否かを判別し（ステップS1704）、存在する場合は、呼び出し元が除外モジュールであるかどうかを判別する（ステップS1705）。OSモジュールリストに存在し、且つ除外モジュールリストに存在しない場合、本ログ取得プログラムは、呼び出し先のモジュールをOSモジュールリストに追加する（ステップS1711）だけで、そのまま元の関数／メソッドを呼び出す（ステップS1712）。

**【0047】**

呼び出し元のモジュールが、ステップS1704においてOSモジュールリストに存在しない場合、及びステップS1705において除外モジュールリストに存在する場合は、DLL名／インターフェース名／関数名／メソッド名／呼び出し時の時刻をHDDに保存し（ステップS1706）、その呼び出しに対してのパラメータをHDDに保存する（ステップS1707）。

**【0048】**

続いて、本体の関数／メソッドを呼び出す（ステップS1708）。関数／メソッド内部の処理が終了すると、DLL名／インターフェース名／関数名／メソッド名／終了時の時刻をHDDに保存し（ステップS1709）、その呼び出しに対してのパラメータ及び戻り値をHDDに保存する（ステップS1710）。続いて、関数／メソッドのリターン処理を行なう（ステップS1713）。

**【0049】**

この処理は、評価対象となるプログラムが終了するまで（ステップS1714

) 続行される。

#### 【0050】

このように「オペレーティング・システムのサーフェイス」及び「サーフェイスからの除外モジュール」を指定可能なA P Iトレーサにより、ソフトウェアに対する適切な処理ログが取得可能となり、バグ原因特定などの解析が可能となる。

#### 【0051】

##### 【第2の実施形態（その3）】

上記第2の実施形態（その1）、（その2）において、図14のステップS1401、図17のステップS1701では、OSモジュールリストの取得をオペレーティング・システムが提供した情報によって自動的に取得することとしたがこれに限られない。また、「オペレーティング・システムが提供した情報」は、オペレーティング・システムが用意したインターフェースから取得するものでも構わないし、例えばWindows（登録商標）OSの場合、レジストリ情報でも構わない。

#### 【0052】

また、図14に示した処理を行なうか、図17に示した処理を行なうかを、ユーザが明示的に選択できるようにしてもよい。

#### 【0053】

さらに、上記第2の実施形態（その2）では、あらかじめオペレーティング・システムの除外モジュールの定義がなされていることとしたが、図17のステップS1702において、ユーザが任意に指定できるようにしてもよい。

#### 【0054】

##### 【第3の実施形態（その1）】

上記第1の実施形態では、関数定義ファイルとして、D L L名／インターフェース名／関数名／メソッド名と、その関数／メソッドに対するパラメータ及び戻り値の型とを記述することとしたが（図10参照）、本発明にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルの記述は、これに限られない。本実施形態では、特殊な構造体データを含む関数として、D E

V MODEを含む関数のログを取得する場合の、関数定義ファイルの記述方法と当該関数定義ファイルにおけるログ取得方法について説明する。

#### 【0055】

DEVMODEとは、プリンタドライバのユーザインターフェースで設定できる印刷設定がどのように設定されているかを示すWindows（登録商標）の構造体のことをいい、DEVMODEにはOSが共通で定義している部分と、プリンタベンダが拡張して使用できる部分（拡張領域）とがある。そして、DEVMODEの最初に「拡張領域サイズ」が指定されている。このように拡張領域のサイズが指定されている場合には、DEVMODE構造体へのポインタがAPIトレーサのパラメータとして入力された場合、DEVMODEの標準領域（サイズ固定）の最後のアドレスから指定サイズ分だけデータ内容を読み出してログとして取得するように、関数定義ファイルに書き込むことで、DEVMODEを含む関数の処理ログを取得することが可能となる。以下に詳細に説明する。

#### 【0056】

図18は、本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルの一例であり、一般的に広く用いられているIDLにより記述されている。本ソフトウェア評価システムに於いては、このIDLをトークン化したタイプライブラリファイルを関数定義ファイルとして使用する。なお、本実施形態は付加バイナリ情報取得のための定義手段を提供することを特徴とするものであり、関数定義ファイルはIDLでなくても構わない。例えば、通常のテキストファイルを用いることも可能であり、XMLなどのマークアップ言語を用いることも可能である。

#### 【0057】

図19は、本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対して付加バイナリ情報取得を指定することで、ポインタ・データの実体を処理ログとして取得すべく、IDLにより記述されている。

#### 【0058】

TESTSTRUCT\_WITHEXTRA構造体の定義に於いて、long

cExtrabinaryDataに対してcustom(PAT\_PARAM\_ATTR\_ID, "struct size extra\_is()")と宣言している(1901)。ここで、PAT\_PARAM\_ATTR\_ID(1900)はIDLの中で本ソフトウェア評価システムが利用するための識別子である。ここで、"struct size extra\_is()"と定義しておき、FuncStructSizeextra\_is関数に引数をTESTSTRUCT\_WITHEXTRA\* lParamと定義しておく(1902)ことにより、FuncStructSizeextra\_is関数が実際に呼び出された際に、TESTSTRUCT\_WITHEXTRA構造体の置かれたメモリ領域の先頭から、cExtrabinaryDataが指すサイズ分だけの、付加情報としてのバイナリデータを処理ログとして保存する。

#### 【0059】

図20は、図19に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

#### 【0060】

この図では、TESTSTRUCT\_WITHEXTRA構造体(2000)が置かれたメモリの先頭アドレスの0x01F7B000であるとして説明する。1byteのデータであるchParam1(2001)は0x01F7B000に置かれ、4byteのデータであるcExtrabinaryDataSize(2002)は0x01F7B001に置かれ、1byteのデータであるchParam2(2003)は0x01F7B005に置かれ、1byteのデータであるchParam3(2004)は0x01F7B006に置かれている。

#### 【0061】

ここまで通常の構造体メモリ配置と同一であるが、TESTSTRUCT\_WITHEXTRA構造体はchParam3に続くメモリ、すなわち0x01F7B007からの領域にExtrabinaryData(2005)という付加データを持っており、そのサイズがcExtrabinaryDataSizeに応じて可変となっている。例えば、cExtrabinaryDataS

izeの値として8が入っている場合には、ExtraBinaryDataは8byte存在することになるため、TESTSTRUCT\_WITHEXTRA構造体の最後尾のメモリアドレスは0x01F7B00Eとなる。このため、TESTSTRUCT\_WITHEXTRA構造体の指し示すすべてのデータをログに取得するためには、このExtraBinaryDataを、cExtraBinaryDataSizeの値の分だけバイナリデータとして取得する必要がある。図19におけるstruct size extra\_isの定義は、この必要を満たすために記述されている。

#### 【0062】

図21は、本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図19に示すように関数定義ファイルが記述されている時の処理ログを取得する場合の処理の流れを示すフローチャートである。

#### 【0063】

処理が開始されると（ステップS2100）、ログ取得を開始し、モジュール名／インターフェース名／関数名／メソッド名をHDDに保存する（ステップS2101）。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存する（ステップS2102）。そして、関数定義にstruct size extra\_isの設定が存在するかどうか、すなわちstruct size extra\_isの定義を含んだ構造体がパラメータに存在するか否かを判別し（ステップS2103）、存在する場合には、まずその定義を含む構造体のサイズを計算する（ステップS2104）。

#### 【0064】

続いて、構造体の先頭ポインタから計算した構造体サイズの分だけずれたメモリ領域から、struct size extra\_isで示されたメンバの値によって示されるサイズ分だけ、データをHDDに保存する（ステップS2105）。続いて、元の関数／メソッドを呼び出す（ステップS2106）。

#### 【0065】

関数／メソッドからリターンすると、ログ取得コードはリターン時の時刻、戻

り値及びポインタ・パラメータの指すメモリの内容をHDDに保存する（ステップS2107）。そして、関数定義にstruct sizeextra\_isの設定が存在するかどうか、すなわちstruct sizeextra\_isの定義を含んだ構造体がパラメータ／戻り値に存在するか判別し（ステップS2108）、存在する場合には、その定義を含む構造体のサイズを計算する（ステップS2109）。続いて、構造体の先頭ポインタから計算した構造体サイズの分だけされたメモリ領域から、struct sizeextra\_isで示されたメンバの値によって示されるサイズ分だけ、データをHDDに保存する（ステップS2110）。この処理はユーザから終了の命令により終了する（ステップS2111）。なお、struct sizeextra\_isで指定されたメンバが複数にわたる場合は、当然ステップS2104～ステップS2105及びステップS2109～ステップS2110の処理が指定された数すべてに対して行なわれる。

#### 【0066】

図22は、本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図19の関数定義ファイルにより取得された処理ログを示す図である。

#### 【0067】

ここでは処理ログはテキスト部分（2200）とバイナリ部分（2201）とに分けて表示しており、テキストデータに書き込まれたDataIDが、バイナリ部分に書き込まれたDataIDに対応している。例えば、一度目の呼び出しのログではDataIDが0x0001であるが、このIDがバイナリ部分に書き込まれた付加バイナリ情報、DataID=0x0001を指し示している。この場合、FuncStructsizeextraIsの1度目の呼び出しでは8バイト、2度目の呼び出しでは40バイト、3度目の呼び出しでは5バイト、4度目の呼び出しでは7バイトの付加バイナリデータが取得されている。struct sizeextra\_isの定義を行なわない場合には、chParam1及びcExtrabinaryDatasize及びchParam2及びchParam3のみが取得されるが、struct sizeextra\_is

の定義を行なうことにより、付加バイナリデータである ExtraBinaryDataを、それぞれの呼び出しに応じたサイズ分、処理ログに取得することが可能となっている。

#### 【0068】

このように、構造体のメンバに応じた処理ログ取得のための関数定義手段を設けることにより、通常では取得しきれないデータを、バイナリデータとして処理ログに取得することが可能となる。

#### 【0069】

##### 【第3の実施形態（その2）】

上記第3の実施形態（その1）では、特定の構造体の最後尾のアドレスから所定サイズ分のデータを処理ログとして取得するための関数定義ファイルの記述方法と、その場合のログ取得プログラムの処理について述べたが、本発明にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルの記述方法はこれに限られない。

#### 【0070】

図23は、本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対してメンバをオフセットとして指定することで、ポインタ・データの実体をログとして取得すべく、IDLにより記述されている。

#### 【0071】

TESTSTRUCT\_WITHPTRDIFF構造体の定義に於いて、LPSString及びlong nNumberに対してcustom(PAT\_PARAM\_ATTR\_ID, "ptrdiff()")と宣言している(2301、2302)。ここで、PAT\_PARAM\_ATTR\_ID(2300)はIDLの中で本ソフトウェア評価システムが利用するための識別子である。ここで、"ptrdiff()"と定義しておき、FuncPtrdiff函数に引数をTESTSTRUCT\_WITHPTRDIFF\* lparamと定義しておく(2303)ことにより、FuncPtrdiff函数が実際に呼び出された際に、TESTSTRUCT\_WITHPTRDIFF構

造体の置かれたメモリ領域の先頭から、`lpszString` 及び `nNumber` の数値によって表されるオフセットだけずれたメモリ領域に格納された、`LPSTR` 及び `long` のデータを処理ログとして保存する。

#### 【0072】

`ptrdiff()` の宣言は、以下の2つの条件定義を兼ねている。

- (1) `ptrdiff()` で指定された引数は、実際には規定された型ではなく、構造体の先頭アドレスに対するオフセットを表す数値として扱われること。
- (2) そのオフセットの指し示すメモリエリアに、規定された型のデータが保持されていること。

#### 【0073】

図24は、図18に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

#### 【0074】

この図では、`TESTSTRUCT_WITHPTRDIFF` 構造体 (240) が置かれたメモリの先頭アドレスの `0x01F7B000` であるとして説明する。`byte` のデータである `chParam1` (2401) は `0x01F7B000` に置かれ、`byte` のデータである `lpszString` へのオフセット (2402) は `0x01F7B001` に置かれ、`byte` のデータである `nNumber` へのオフセット (2403) は `0x01F7B005` に置かれ、`byte` のデータである `chParam2` (2404) は `0x01F7B009` に置かれている。`TESTSTRUCT_WITHPTRDIFF` 構造体において、`lpszString` の実体はオフセット (2402) によって示されたサイズ分、構造体の先頭アドレスからずらした領域に格納されており (2405)、`nNumber` の実体はオフセット (2403) によって示されたサイズ分、構造体の先頭アドレスからずらした領域に格納されている (2406)。このため、`TESTSTRUCT_WITHPTRDIFF` 構造体の指し示すすべてのデータを処理ログに取得するためには、これら `lpszString` 及び `nNumber` に関する、実体のデータをログとして取得する必要がある。図23における `ptrdiff` の定義は、この必要を満たすために記述されている。

**【0075】**

図25は、本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図23に示すように関数が定義されている時のログを取得する場合の処理の流れを示すフローチャートである。

**【0076】**

処理が開始されると（ステップS2501）、ログ取得を開始し、モジュール名／インターフェース名／関数名／メソッド名をHDDに保存する（ステップS2502）。次にログ取得コードは、呼び出し時の時刻、パラメータ及びポインタ・パラメータの指すメモリの内容をHDDに保存する（ステップS2503）。そして、関数定義ファイルにptrdiffの設定が存在するかどうか、すなわちptrdiffの定義を含んだ構造体がパラメータに存在するか否かを判別し（ステップS2504）、存在する場合には、構造体の先頭アドレスとptrdiffで定義された構造体メンバのオフセットの値を合算し、ptrdiffで定義された構造体メンバがどのアドレスに格納されているかを計算する（ステップS2505）。続いて、計算したアドレスの指示するメモリ領域から、ptrdiffで示されたメンバの型のデータをHDDに保存する（ステップS2506）。続いて、元の関数／メソッドを呼び出す（ステップS2507）。関数／メソッドからリターンすると、ログ取得コードはリターン時の時刻、戻り値及びポインタ・パラメータの指すメモリの内容をHDDに保存する（ステップS2508）。そして、関数定義ファイルにptrdiffの設定が存在するかどうか、すなわちptrdiffの定義を含んだ構造体がパラメータ／戻り値に存在するか否かを判別し（ステップS2509）、存在する場合には、構造体の先頭アドレスとptrdiffで定義された構造体メンバのオフセットの値を合算し、ptrdiffで定義された構造体メンバがどのアドレスに格納されているかを計算する（ステップS2510）。続いて、計算したアドレスの指示するメモリ領域から、ptrdiffで示されたメンバの型のデータをHDDに保存する（ステップS2511）。この処理はユーザから終了の命令により処理を終了する（ステップS2512）。なお、ptrdiffで指定されたメンバが複数に渡る場合は、当然ステップS2505～ステップS2506及びステップS25

10～ステップS2511の処理が指定された数すべてに対して行なわれる。

#### 【0077】

図26は、本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図23の関数定義ファイルにより取得された処理ログを示す図である。それぞれの呼び出しでオフセットが異なっているが、本ソフトウェア評価システムは、オフセットで指示された実体を的確に処理ログとして取得している。

#### 【0078】

以上の説明から明らかなように、本実施形態によれば、構造体のメンバに応じたログデータ取得のための定義手段を設けることにより、通常では取得しきれないデータを含む関数／メソッドであっても、処理ログを取得することが可能となる。

#### 【0079】

##### 【第4の実施形態（その1）】

上記第1の実施形態では、関数定義ファイルとして、DLL名／インターフェース名／関数名／メソッド名と、その関数／メソッドに対するパラメータ及び戻り値の型とを記述することとしたが（図10参照）、本発明にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルの記述は、これに限られない。本実施形態では、特殊なCOMに対してログを取得する場合の、関数定義ファイルの記述方法について説明する。

#### 【0080】

一般に、プリンタドライバには、OSに予め組み込まれているUNIDRIVER（ユニドライバ）を用いる場合と、モノレシックドライバを用いる場合がある。そして、プリンタベンダーはモノレシックドライバ、またはユニドライバからの出力を受け取るプラグインモジュール（\*.drv）を提供している。モノレシックドライバの場合は、GDI32.dllは一般的なDDI関数を用いる。

#### 【0081】

ここで、DLL同士は、レジストリに登録されているAPIを用いてデータの

やり取りを行うが、ユニドライバがプリントベンダが提供するプラグインモジュールに出力する処理内容は、レジストリに登録されていないCOM（object-AP）である。このため、上記第1の実施形態において示した関数定義ファイルでは、特殊なCOMについての処理ログを取得することができない。そこで、本実施形態ではユニドライバが吐き出すレジストリに登録されていないCOMを通常のCOMとして扱うように関数定義ファイルを記述することで適切なログを取得することとする。

### 【0082】

図27は、本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、それぞれの関数／メソッドのパラメータの形式や、戻り値の形式を指示する関数定義ファイルの例であり、一般的に広く用いられているIDLにより記述されている。DL名／インターフェース名／関数名／メソッド名を記述し、その関数／メソッドに対する、パラメータ／戻り値の型が示されている。本発明の第4の実施形態にかかるソフトウェア評価システムに於いては、このIDLをトークン化したタイプライブラリファイルを関数定義ファイルとして使用する。なお、本実施形態は、より詳細な情報取得のための関数定義手段を提供することを特徴とするものであり、関数定義ファイルはIDLにより記述されていなくても構わない。例えば、通常のテキストファイルを用いることも可能であり、XMLなどのマークアップ言語を用いることも可能である。

### 【0083】

本発明の第4の実施形態にかかるソフトウェア評価システムは、この関数定義ファイルによって指示された内容を元に、それぞれの関数／メソッドがどのようなパラメータ／戻り値を有しているかを判断し、その内容をログとして取得する。なお、本実施形態では、Interface testに実装されたDIGetcClassObjectという名前の関数が、以下のようなパラメータを持つ状態を例としている。

- (1) 入力パラメータの"rclsid"、すなわち「クラスID」をあらわすパラメータ。
- (2) 入力パラメータの"rid"、すなわち「リファレンス・インターフェー

ス I D」をあらわすパラメータ。

(3) 出力パラメータの”`ppv`”、すなわち、上記2つの入力パラメータを元に得られた、実体としてのインターフェースポインタ。

#### 【0084】

DLL内にコードとして記述された`Interface`内のメソッドを、EXEが呼び出すためには、先に`DllGetClassObject`関数を用いて、クラスID及びリファレンス・インターフェースIDを元にした`Interface`のインスタンスを生成し、そのインスタンスのメモリアドレスを知っておく必要がある。

#### 【0085】

図27に示された”`iid_is`”(2703)が、本実施形態の特徴を最もよくあらわすものである。すなわち、`iid_is`に第二引数である`rid`を指定することで「この出力パラメータは、実際には`void *`ではなく、`rid`に指定された型のインターフェースである」という定義を行なっている。この定義をAPIトレーサが認識することにより、従来取得できなかった「`ppv`で返ってきたインターフェース内に実装されたメソッドの呼び出し」に関しても、詳細が処理ログとして取得できるようになっている。

#### 【0086】

図28は、`rid`に指定するインターフェース型を定義した、関数定義ファイルの例である。ここでは、`IUnknown`という名前のインターフェースから派生した、`IGetInfo`というインターフェースを定義している。`IGetInfo`には、`GetName`及び`FreeNameBuffer`という2つのメソッドが実装されている。

#### 【0087】

図29は、本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

#### 【0088】

処理が開始されると（ステップS2900）、設定された関数／メソッドが呼び出される都度に、DLL名／インターフェース名／関数名／メソッド名／呼び

出し時の時刻をHDDに保存し（ステップS2901）、その呼び出しに対してのパラメータをHDDに保存する（ステップS2902）。続いて、入力パラメータの関数定義ファイルに、図27によって説明した*i i d\_i s*、すなわち「リファレンス・インターフェースIDの定義」があるかどうかを判別し（ステップS2903）、存在する場合は、そのリファレンス・インターフェースIDを、メモリ上の別のエリアに保存しておく（ステップS2904）。

#### 【0089】

本体の関数／メソッドを呼び出し（ステップS2905）、メソッド内部の処理が終了すると、DLL名／インターフェース名／関数名／メソッド名／終了時の時刻をHDDに保存し（ステップS2906）、その呼び出しに対してのパラメータ及び戻り値をHDDに保存する（ステップS2907）。

#### 【0090】

次に、出力パラメータがインターフェースへのポインタとして定義されているかどうかを、図27に示した関数定義ファイルに基づいて判断し（ステップS2908）、定義されている場合は、リファレンス・インターフェースIDがメモリ上に保存されているかどうかを判断する（ステップS2909、ステップS2910）。リファレンス・インターフェースIDが保存されていない場合には、その出力パラメータをインターフェースそのものへのポインタとして認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップS2911）、該当するパラメータ／戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップS2912）。リファレンス・インターフェースIDが保存されている場合、その出力パラメータを、リファレンス・インターフェースIDがあらわすインターフェースへ派生した形のポインタ、例えば図28に示した*I G e t I n f o*として認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップS2913）、該当するパラメータ乃至戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップS2914）。

#### 【0091】

ステップS2912またはステップS2914の処理が終了するか、もしくはステップS2908においてインターフェースポインタとして定義されていない

ことを判断した場合は、関数／メソッドのリターン処理を行なう（ステップS2915）。

#### 【0092】

この処理は、評価対象となるプログラムが終了するまで（ステップS2916）続行される。

#### 【0093】

図30は、図29における処理によって取得された処理ログの例である。図27におけるi id\_isの定義により、D11GetClassObjectの出力パラメータppvをI GetInfo型のインターフェースとして認識した結果、D11GetClassObjectのログだけでなく、I GetInfoインターフェース内に実装されたメソッドの呼び出しに関しても、処理ログとして取得することが可能となっている。なお本実施形態は、リファレンス・インターフェースIDを認識することを特徴とするものであり、実際のリファレンス・インターフェースID（3001）とI GetInfoとの関連付けは、公知の技術を元に行なわれる。例えば、Windows（登録商標）OSにおいては、この情報はレジストリに置かれており、レジストリ情報を元に関連付けが可能である。ただし、他の方法を用いて関連付けを行なっても構わない。

#### 【0094】

以上の説明から明らかなように、本実施形態によれば、関数定義ファイルとして、「ある入力パラメータが、出力パラメータの型をあらわす」ことを指定することにより、ソフトウェアに対する更に詳細な処理ログを取得することが可能となり、バグ原因特定など、より詳細な解析が可能となる。

#### 【0095】

##### 【第4の実施形態（その2）】

上記第4の実施形態（その1）においては、実体すなわちインスタンスとしてのインターフェースの解決を行なっているが、これだけでは「インスタンスを生成する元となっている情報」すなわち、オブジェクト指向プログラミングにおける「クラス情報」に基づいた情報を、処理ログとして取得できない。そこで本実施形態では、クラス情報を追加情報として処理ログに取得するために、クラスI

DをインターフェースIDと関連付ける場合について説明する。

#### 【0096】

本実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルでは、クラスIDは、図27におけるD11GetClassObjectのrc1s\_idをもって定義し、その定義のために「clsid\_is」という関数定義を用いる。この「clsid\_is」とは、「このD11GetClassObjectにおいては、clsid\_isで指定されたパラメータをクラスIDとするクラス情報を元にしてインターフェースのインスタンスが生成されている」ということを指定するための関数定義である。このように関数定義ファイルを記述することにより、本実施形態にかかるソフトウェア評価システムは、より詳細な情報をログとして取得することが可能となっている。

#### 【0097】

図31及び図32は、本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートであり、図29とは別の一例である。

#### 【0098】

処理が開始されると（ステップS3100）、設定された関数／メソッドが呼び出される都度に、DLL名／インターフェース名／関数名／メソッド名／呼び出し時の時刻をHDDに保存し（ステップS3101）、その呼び出しに対するパラメータをHDDに保存する（ステップS3102）。続いて、入力パラメータの関数定義ファイルに、図27及び上記によって説明したclsid\_is、すなわち「クラスIDの定義」があるかどうかを判別し（ステップS3103）、存在する場合は、そのクラスIDをメモリ上の別のエリアに保存しておく（ステップS3104）。続いて、入力パラメータの関数定義ファイルに、図27によって説明したiid\_is、すなわち「リファレンス・インターフェースIDの定義」があるかどうかを判別し（ステップS3105）、存在する場合は、そのリファレンス・インターフェースIDをメモリ上の別のエリアに保存しておく（ステップS3106）。

#### 【0099】

本体の関数／メソッドを呼び出し（ステップS3107）、メソッド内部の処理が終了すると、D L L名／インターフェース名／関数名／メソッド名／終了時の時刻をH D Dに保存し（ステップS3108）、その呼び出しに対してのパラメータ／戻り値をH D Dに保存する（ステップS3109）。次に、出力パラメータがインターフェースへのポインタとして定義されているかどうかを、図27に示した関数定義ファイルに基づいて判断し（ステップS3110）、定義されている場合は、リファレンス・インターフェース I Dがメモリ上に保存されているかどうかを判断する（図32のステップS3200、S3201）。リファレンス・インターフェース I Dが保存されていない場合、その出力パラメータを、インターフェースそのものへのポインタとして認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップS3203）、該当するパラメータ／戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップS3204）。リファレンス・インターフェース I Dが保存されている場合、その出力パラメータをリファレンス・インターフェース I Dがあらわすインターフェースへ派生した形のポインタ、例えば図28に示したI G e t I n f oとして認識し、メモリ上にそのポインタに対するログ取得コードを生成して（ステップS3205）、該当するパラメータ／戻り値を、ログ取得コードのメモリアドレスに書き換える（ステップS3206）。

#### 【0100】

ステップS3204またはステップS3206の処理が終了すると、メモリにクラス I Dが保存されているかを判別し（ステップS3207）、保存されている場合は、保存されたクラス I Dをログ取得コードと関連付けてメモリに保持しておく（ステップS3208）。続いて、関数／メソッドのリターン処理を行なう（図31のステップS3112）。この処理は、評価対象となるプログラムが終了するまで（ステップS3113）続行される。

#### 【0101】

ステップS3208においてクラス I Dを保持しておくことにより、I G e t I n f o内の関数が呼ばれた際に、そのクラス情報を元に、ステップS3101においてより詳細な情報をログに取得することが可能となる。図33は、図30

に対して、クラスIDを元にモジュール名が追加でログ取得可能となっていることをあらわしている（3303、3304、3305、3306）。

#### 【0102】

なお本実施形態は、クラスIDをインターフェース・パラメータと関連付けつつ、リファレンス・インターフェースIDを認識することを特徴とするものであり、実際のクラスID（3000）と、モジュール名等詳細情報との関連付けは、公知の技術を元に行なわれる。例えば、Windows（登録商標）OSにおいては、この情報はレジストリに置かれており、レジストリ情報を元に関連付けが可能である。が、他の方法を用いて関連付けを行なっても構わない。

#### 【0103】

以上の説明から明らかなように、本実施形態によれば、関数定義ファイルにおいて「ある入力パラメータが、出力パラメータの型をあらわす」ことを指定することにより、ソフトウェアに対する更に詳細な処理ログを取得することが可能となり、バグ原因特定など、より詳細な解析が可能となる。

#### 【0104】

##### 【第5の実施形態（その1）】

上記第4の実施形態によれば、特殊なCOMに対するログを取得することが可能となるが、かかる特殊なCOMの場合、処理ログの対象となるモジュール名がレジストリに定義されていないため、モジュール名に基づいて処理ログを取得することができない。そこで、本実施形態では個々のインターフェース名（COMの処理内容）に対するモジュール名を追加定義として用意し、処理ログを作成する際にインターフェース名からモジュール名を判断して書き込むこととする。

#### 【0105】

図34は、本発明の第5の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける関数定義ファイルの一例であり、一般的に広く用いられているIDLにより記述されている。本ソフトウェア評価システムに於いては、このIDLをトークン化したタイプライブラリ・ファイルを関数定義ファイルとして使用する。

#### 【0106】

IDLはインターフェースの定義に用いられるものであり、標準ではモジュール名の定義は不可能である。例えば、複数のモジュールが、同じインターフェース定義でインターフェースを提供しており、それらが選択的にプラグイン・モジュールとして用いられている場合、インターフェース定義を行なっても、それらのインターフェースがどのプラグイン・モジュールに属するものであるかは、それらプラグイン・モジュールを選択的に呼び出しているモジュールのみが知り得る。例えば、オペレーティング・システムが選択的にドライバ用のプラグイン・モジュールを呼び出している場合、オペレーティング・システムが選択を行なうアルゴリズムが公開されていない場合は、モジュール名を関連付けて呼び出しに関する詳細情報を処理ログに取得することは不可能である。具体的には、Microsoft Windows（登録商標）オペレーティング・システムに搭載されている、ユニバーサル・プリンタドライバのプラグイン・モジュールなどが考えられる。

#### 【0107】

ただし、これらのプラグイン・モジュールは、ユーザの操作に関連付けられて選択的に呼び出されているケースが多い。例えば、プリンタドライバにおいては、複数搭載されたプリンタのうち、ユーザがどのプリンタで出力するのかを選択して、その選択操作に対して1対1に対応するプラグイン・モジュールが呼び出される。この場合、IDLによるインターフェース定義とは別途「今回の操作に対してはこのモジュールが呼び出される」ということを知り得て、尚且つその情報を、ログ取得プログラムに対し指示する手段があれば、その手段を用いて、上記プラグイン・モジュールのモジュール名も含めた形の処理ログを取得することが可能となる。

#### 【0108】

図35は、ライブラリ名に対してモジュール名を定義した関数定義ファイルの例であり、3500でライブラリ名を、3501でモジュールの置かれているフォルダ名を、3502でモジュール名をそれぞれ定義している。

#### 【0109】

図36は、インターフェース名に対してモジュール名を定義した関数定義ファ

イルの例であり、3600でインターフェース名を、3601でモジュールの置かれているフォルダ名を、3602でモジュール名をそれぞれ定義している。

#### 【0110】

なお、本実施形態では、図35乃至図36におけるモジュール名の追加定義を、単なるテキストファイルで示しているが、本実施形態はIDLのような「DL/L/インターフェース/関数/メソッドを定義するための標準フォーマット」に対して別のフォーマットの追加定義を行ない、それを元に関数/メソッドのログを取得することを特徴とするものであり、追加定義は単なるテキストファイルではなく、HTMLやXMLといったマークアップ言語で記述されていても構わない。

#### 【0111】

図37は、本発明の第5の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートであり、本実施形態の特徴を最もよくあらわすものである。

#### 【0112】

処理が開始されると（ステップS3700）、設定された関数/メソッドが呼び出される都度に、DLL名/インターフェース名/関数名/メソッド名/呼び出し時の時刻をHDDに保存し（ステップS3701）、その呼び出しに対するパラメータをHDDに保存する（ステップS3702）。続いて、入力パラメータの関数定義ファイルに、図35乃至図36によって説明した、モジュール名の追加定義が存在するかどうかを判別し（ステップS3703）、存在する場合は、そのモジュール名をメモリ上の別のエリアに保存しておく（ステップS3704）。

#### 【0113】

本体の関数/メソッドを呼び出し（ステップS3705）、メソッド内部の処理が終了すると、DLL名/インターフェース名/関数名/メソッド名/終了時の時刻をHDDに保存し（ステップS3706）、その呼び出しに対するパラメータ及び戻り値をHDDに保存する（ステップS3707）。続いて、関数/メソッドのリターン処理を行なう（ステップS3708）。

### 【0114】

この処理は、評価対象となるプログラムが終了するまで（ステップS3709）続行される。

### 【0115】

このように、モジュール名を関数定義とは別のフォーマットで追加定義して、それを元に処理ログの取得を行なうことにより、例えばプラグイン・モジュールがユーザの操作に関連付けられて選択的に呼び出されているケースに対して、プラグイン・モジュールのモジュール名も含めた形の処理ログを取得することが可能となる。この機能を持つA P I トレーサにより、ソフトウェアに対してモジュール名を必ず含んだ形での処理ログが取得可能となり、バグ原因特定など、より詳細な解析が可能となる。

### 【0116】

#### 【第5の実施形態（その2）】

上記第5の実施形態（その1）においては、モジュール名の追加定義をテキストファイルで行なっているが、これに限らず、モジュール定義をユーザが選択できるようなユーザー・インターフェースを備えてもよい。実現方法としては、図35乃至図36におけるP a t h 及びM o d u l e N a m e の設定を、ユーザが選択可能なユーザー・インターフェースを備えても構わないし、全体のモジュール名追加定義テキストファイルを複数用意して、それらのファイルを選択するユーザ・インターフェースを備えても構わない。

### 【0117】

#### 【第5の実施形態（その3）】

上記第5の実施形態（その1）および（その2）においては、モジュール名の追加定義を、ユーザが指定乃至選択しているが、これに限らず、追加のモジュール定義をオペレーティング・システムが備えるインターフェースから取得しても構わない。例えば、M i c r o s o f t の. N E T F r a m e w o r k では、各モジュールがXML形式のモジュール定義データを含み、モジュール定義データをオペレーティング・システムのインターフェースで取得できる。また、この場合に、オペレーティング・システムから取得したモジュール定義に示された

モジュールに対して、インポート関数アドレステーブル及びエクスポート関数アドレステーブルなどの、モジュール内部に用意されたヘッダ情報を元に、メモリ上でそのモジュール内に実装された関数／メソッドをログ取得ルーチンにリダイレクトして、更にログ情報を詳細に取得することも可能である。例えば、MicrosoftのWindows（登録商標）オペレーティング・システムでは、この機能によって、通常のDLLに実装されたエクスポート関数の呼び出し、COMインターフェースに実装されたメソッドの呼び出し、.NETアセンブリに実装されたメソッドの呼び出しの3点について、1つのログデータ内に時系列ですべての情報を並べることが可能である。

### 【0118】

#### 【他の実施形態】

なお、本発明は、複数の機器（例えばホストコンピュータ、インターフェイス機器、リーダ、プリンタなど）から構成されるシステムに適用しても、一つの機器からなる装置（例えば、複写機、ファクシミリ装置など）に適用してもよい。

### 【0119】

また、本発明の目的は、前述した実施形態の機能を実現するソフトウェアのプログラムコードを記録した記憶媒体を、システムあるいは装置に供給し、そのシステムあるいは装置のコンピュータ（またはCPUやMPU）が記憶媒体に格納されたプログラムコードを読み出し実行することによっても、達成されることは言うまでもない。

### 【0120】

この場合、記憶媒体から読み出されたプログラムコード自身が前述した実施形態の機能を実現することになり、そのプログラムコードを記憶した記憶媒体は本発明を構成することになる。

### 【0121】

プログラムコードを供給するための記憶媒体としては、例えば、フロッピ（登録商標）ディスク、ハードディスク、光ディスク、光磁気ディスク、CD-ROM、CD-R、磁気テープ、不揮発性のメモリカード、ROMなどを用いることができる。

**【0122】**

また、コンピュータが読出したプログラムコードを実行することにより、前述した実施形態の機能が実現されるだけでなく、そのプログラムコードの指示に基づき、コンピュータ上で稼働しているOS（オペレーティングシステム）などが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

**【0123】**

さらに、記憶媒体から読出されたプログラムコードが、コンピュータに挿入された機能拡張ボードやコンピュータに接続された機能拡張ユニットに備わるメモリに書込まれた後、そのプログラムコードの指示に基づき、その機能拡張ボードや機能拡張ユニットに備わるCPUなどが実際の処理の一部または全部を行い、その処理によって前述した実施形態の機能が実現される場合も含まれることは言うまでもない。

**【0124】**

以下に本発明の実施態様の例を示す。

**【0125】**

〔実施態様1〕 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内の関数のうち、指定された関数を識別する工程と、

ロードされた前記所定の処理を行う関数のアドレスと前記指定されたオペレーティングシステム内の関数のアドレスとを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と  
を備えることを特徴とするログ取得方法。

#### 【0126】

【実施態様2】 前記所定の処理を行う関数および前記指定されたオペレーティングシステム内の関数を呼び出す際の所定の情報は、少なくとも、呼び出された関数の関数名、呼び出す際の時刻、呼び出す際のパラメータ、または呼び出す際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様1に記載のログ取得方法。

#### 【0127】

【実施態様3】 前記実行結果を受け取った際の所定の情報は、少なくとも、受け取った際の時刻、受け取った際のパラメータ、受け取った際の戻り値、または受け取った際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様1に記載のログ取得方法。

#### 【0128】

【実施態様4】 前記所定の処理を行う関数のアドレスは、該関数を提供するダイナミックリンクライブラリごとに、インポート関数アドレステーブルに記載されていることを特徴とする実施態様1に記載のログ取得方法。

#### 【0129】

【実施態様5】 前記所定の処理を行う関数のアドレスのうち、前記ログ取得のための関数のアドレスに書き換えるアドレスを選択する工程を更に備え、

前記書き換える工程は、前記選択する工程により選択された関数のアドレスを書き換えることを特徴とする実施態様1に記載のログ取得方法。

#### 【0130】

【実施態様6】 所定の1または複数のモジュールを定義する工程と、前記関数の呼び出しが当該定義したモジュールを経由したか否かを判別する工程とを更に備え、当該定義されたモジュールを経由した場合には、当該関数を呼び出す際の前記所定の情報を記録対象から除外することを特徴とする実施態様1のログ取得方法。

#### 【0131】

〔実施態様7〕 オペレーティング・システムに含まれる所定の1または複数のモジュールを定義する工程と、前記関数の呼び出しが経由したモジュールを判別する工程とを更に備え、当該定義されたモジュール以外のオペレーティング・システムに含まれるモジュールを経由した場合には、当該関数を呼び出す際の前記所定の情報を記録対象から除外することを特徴とする実施態様1のログ取得方法。

#### 【0132】

〔実施態様8〕 前記所定の情報が構造体である場合に、当該構造体が置かれたメモリ領域の先頭から当該構造体のサイズ分だけ後のメモリ領域に格納されている情報を、当該構造体に指定されたサイズに応じて記録することを特徴とする実施態様1のログ取得方法。

#### 【0133】

〔実施態様9〕 前記所定の情報が構造体である場合に、当該構造体が置かれたメモリ領域の先頭から当該構造体に対して指定されたオフセットより後のメモリ領域に格納されている情報を、定義されたデータ型のデータとして記録することを特徴とする実施態様1のログ取得方法。

#### 【0134】

〔実施態様10〕 何れのモジュールの何れの関数の所定の情報の何れがオブジェクトの情報を表わし、何れがオブジェクトのクラスIDを表わすかを定義しておき、関数の呼び出しにおいて、オブジェクトのクラスIDに基づいて、その関数の所定の情報に格納されたオブジェクトの情報を記録することを特徴とする実施態様1のログ取得方法。

#### 【0135】

〔実施態様11〕 何れのモジュールの何れの関数の所定の情報の何れがオブジェクトの情報を表わし、何れがオブジェクトのインターフェースIDを表わすかを定義しておき、関数の呼び出しにおいて、オブジェクトのインターフェースIDに基づいて、その関数の所定の情報に格納されたオブジェクトの情報を記録することを特徴とする実施態様1のログ取得方法。

#### 【0136】

〔実施態様12〕 ライブドアリまたは各インターフェースに対するモジュール名の追加定義を用意しておき、関数を呼び出す際に、当該追加定義を参照してモジュール名を記録することを特徴とする実施態様1のログ取得方法。

#### 【0137】

〔実施態様13〕 所定の処理を行う関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記所定の処理を行う関数がロードされた場合のアドレスから所定のオフセットを有する領域であって、指定された領域を識別する工程と、

前記ロードされた前記所定の処理を行う関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記所定の処理を行う関数を呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行う関数を呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と、

前記指定された領域のデータを読み出して記録する工程と

を備えることを特徴とするログ取得方法。

#### 【0138】

〔実施態様14〕 所定の処理を行う第1の関数を備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に該プログラムの実行ファイルから直接呼び出された前記第1の関数によってのみ呼び出される第2の関数のうち、指定された第2の関数を識別する工程と、

前記第1の関数によって呼び出されることでロードされる前記指定された第2の関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程と、を備え、

前記ログ取得のための関数は、

前記指定された第2の関数を呼び出し、該所定の処理を実行させ、実行結果を受け取る工程と、

前記指定された第2の関数を呼び出す際の所定の情報を記録する工程と、  
前記実行結果を受け取った際の所定の情報を記録する工程と  
を備えることを特徴とするログ取得方法。

### 【0139】

【実施態様15】 所定の処理を行う第1の関数を備えるプログラムの実行  
中のログを取得するログ取得方法であって、

前記プログラム実行時に、該プログラムの実行ファイルから直接呼び出された  
前記第1の関数によってのみ呼び出される第2の関数に設定されたIDを識別す  
る工程と、

前記第1の関数によって呼び出されることでロードされる前記IDが設定され  
た第2の関数のアドレスを、ログ取得のための関数のアドレスに書き換える工程  
と、を備え、

前記ログ取得のための関数は、

前記IDが設定された第2の関数を呼び出し、該所定の処理を実行させ、実行  
結果を受け取る工程と、

前記IDが設定された第2の関数を呼び出す際の所定の情報を、前記設定され  
たIDとともに記録する工程と、

前記実行結果を受け取った際の所定の情報を、前記設定されたIDとともに記  
録する工程と

を備えることを特徴とするログ取得方法。

### 【0140】

【実施態様16】 所定の処理を行うメソッドを備えるプログラムの実行中  
のログを取得するログ取得方法であって、

前記プログラム実行時に呼び出されるオペレーティングシステム内のメソッド  
のうち、指定されたメソッドを識別する工程と、

ロードされた前記所定の処理を行うメソッドのアドレスと前記指定されたオペ  
レーティングシステム内のメソッドのアドレスとを、ログ取得のためのメソッド  
のアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドおよび前記指定されたオペレーティングシステム内のメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行うメソッドおよび前記指定されたオペレーティングシステム内のメソッドを呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程とを備えることを特徴とするログ取得方法。

#### 【0141】

[実施態様17] 前記所定の処理を行うメソッドおよび前記指定されたオペレーティングシステム内のメソッドを呼び出す際の所定の情報は、少なくとも、呼び出されたメソッドのメソッド名、呼び出す際の時刻、呼び出す際のパラメータ、または呼び出す際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様16に記載のログ取得方法。

#### 【0142】

[実施態様18] 前記実行結果を受け取った際の所定の情報は、少なくとも、受け取った際の時刻、受け取った際のパラメータ、受け取った際の戻り値、または受け取った際のポインタ・パラメータの指すメモリ内容のいずれかを備えることを特徴とする実施態様16に記載のログ取得方法。

#### 【0143】

[実施態様19] 前記所定の処理を行うメソッドのアドレスは、該メソッドを提供するインターフェースごとに、仮想アドレステーブルに記載されていることを特徴とする実施態様16に記載のログ取得方法。

#### 【0144】

[実施態様20] 前記所定の処理を行うメソッドのアドレスのうち、前記ログ取得のためのメソッドのアドレスに書き換えるアドレスを選択する工程を更に備え、

前記書き換える工程は、前記選択する工程により選択されたメソッドのアドレスを書き換えることを特徴とする実施態様16に記載のログ取得方法。

#### 【0145】

〔実施態様21〕 所定の処理を行うメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

前記所定の処理を行うメソッドがロードされた場合のアドレスから所定のオフセットを有する領域であって、指定された領域を識別する工程と、

前記ロードされた前記所定の処理を行うメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記所定の処理を行うメソッドを呼び出し、該所定の処理を実行させ、受け取った実行結果を前記プログラムに渡す工程と、

前記所定の処理を行うメソッドを呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と、

前記指定された領域のデータを読み出して記録する工程と

を備えることを特徴とするログ取得方法。

#### 【0146】

〔実施態様22〕 所定の処理を行う第1のメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に該プログラムの実行ファイルから直接呼び出された前記第1のメソッドによってのみ呼び出される第2のメソッドのうち、指定された第2のメソッドを識別する工程と、

前記第1のメソッドによって呼び出されることでロードされる前記指定された第2のメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記指定された第2のメソッドを呼び出し、該所定の処理を実行させ、実行結果を受け取る工程と、

前記指定された第2のメソッドを呼び出す際の所定の情報を記録する工程と、

前記実行結果を受け取った際の所定の情報を記録する工程と

を備えることを特徴とするログ取得方法。

#### 【0147】

〔実施態様23〕 所定の処理を行う第1のメソッドを備えるプログラムの実行中のログを取得するログ取得方法であって、

前記プログラム実行時に、該プログラムの実行ファイルから直接呼び出された前記第1のメソッドによってのみ呼び出される第2のメソッドに設定されたIDを識別する工程と、

前記第1のメソッドによって呼び出されることでロードされる前記IDが設定された第2のメソッドのアドレスを、ログ取得のためのメソッドのアドレスに書き換える工程と、を備え、

前記ログ取得のためのメソッドは、

前記IDが設定された第2のメソッドを呼び出し、該所定の処理を実行させ、実行結果を受け取る工程と、

前記IDが設定された第2のメソッドを呼び出す際の所定の情報を、前記設定されたIDとともに記録する工程と、

前記実行結果を受け取った際の所定の情報を、前記設定されたIDとともに記録する工程と

を備えることを特徴とするログ取得方法。

#### 【0148】

〔実施態様24〕 実施態様1乃至23のいずれか1つに記載のログ取得方法をコンピュータによって実現させるための制御プログラム。

#### 【0149】

〔実施態様25〕 実施態様24に記載の制御プログラムを格納した記憶媒体。

#### 【0150】

#### 【発明の効果】

以上説明したように本発明によれば、複数にモジュール分けされたソフトウェアの処理ログを容易に取得でき、かつ、ソフトウェアの障害の原因の解析のための工数を削減することが可能となる。

#### 【図面の簡単な説明】

#### 【図1】

本発明の第1の実施形態にかかるログ取得方法を実現するコンピュータ（ソフトウェア評価システム）の構成をあらわす図である。

【図2】

本発明の第1の実施形態にかかるログ取得方法を説明するためにあらわした、関数ロード時の通常のメモリ構成をあらわす図である。

【図3】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのI A T P a t c h 使用時のメモリ構成をあらわす図である。

【図4 A】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのI A T P a t c h を使用時の状態を示す図である。

【図4 B】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのログ取得処理の流れを示す図である。

【図5】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのI A T P a t c h を使用時の内部構成をあらわす図である。

【図6】

本発明の第1の実施形態にかかるログ取得方法を説明するためにあらわした、C O Mサーバのインターフェースのインスタンス作成時の通常のメモリ構成をあらわす図である。

【図7】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのV T a b l e P a t c h 使用時のメモリ構成をあらわす図である。

【図8 A】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価システムのV T a b l e P a t c h を使用時の状態を示す図である。

【図8 B】

本発明の第1の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

ステムのログ取得処理の流れを示す図である。

【図 9】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの、内部構成をあらわす図である。

【図 10】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムに対して、それぞれの関数及びメソッドのパラメータの形式や、戻り値の形式を指示する、関数定義ファイルの例を示す図である。

【図 11】

本発明の第 1 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで取得した、ログの一例を示す図である。

【図 12】

一般的なソフトウェアのモジュール構成の一例を示す図である。

【図 13】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、処理ログの収集対象となるオペレーティング・システムのモジュールを定義する例を示す図である。

【図 14】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【図 15 A】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムで取得した処理ログの一例を示す図である。

【図 15 B】

OS部分のDLL内の関数が呼び出された際にも、処理ログを取得することとした場合の処理ログの例を示す図である。

【図 16】

本発明の第 2 の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、処理ログの収集対象からオペレーティング・システムの除外モ

ジユールを定義する例を示す図である。

【図17】

本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【図18】

本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例を示す図であり、一般的に広く用いられているIDLにより、記述された図である。

【図19】

本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対して付加バイナリ情報取得を指定することで、ポインタ・データの実体をログとして取得すべく、IDLにより記述したファイルを示す図である。

【図20】

図19に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

【図21】

本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図19に示すように関数が定義されている時のログを取得する場合の処理の流れを示すフローチャートである。

【図22】

本発明の第3の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図19の定義により取得されたログデータを示す図である。

【図23】

本発明の一実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義ファイルを示す図であり、構造体へのポインタに対してメンバをオフセットとして指定することで、ポインタ・データの実体をログとして取得すべく、IDLにより記述したファイルを示す図である。

【図24】

図18に示された関数定義による構造体が、どのようにメモリ上に配置されるかを表す図である。

#### 【図25】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図18に示すように関数が定義されている時のログを取得する場合の処理の流れを示すフローチャートである。

#### 【図26】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図18の定義により取得されたログデータを示す図である。

#### 【図27】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムに対して、それぞれの関数及びメソッドのパラメータの形式や、戻り値の形式を指示する関数定義ファイルの例を示す図である。

#### 【図28】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、`r i i d`に指定するインターフェース型を定義した、関数定義ファイルの例を示す図である。

#### 【図29】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

#### 【図30】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図29における処理によって取得された処理ログの例を示す図である。

#### 【図31】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

#### 【図32】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価シ

システムにおける処理の流れを示すフローチャートである。

【図33】

本発明の第4の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、図31および図32における処理によって取得された処理ログの例を示す図である。

【図34】

本発明の第5の実施形態にかかるログ取得方法を実現するソフトウェア評価システムの関数定義の一例を示す図である。

【図35】

本発明の第5の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、ライブラリ名に対してモジュール名を定義する例を示す図である。

【図36】

本発明の第5の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおいて、インターフェース名に対してモジュール名を定義する例を示す図である。

【図37】

本発明の第5の実施形態にかかるログ取得方法を実現するソフトウェア評価システムにおける処理の流れを示すフローチャートである。

【符号の説明】

- 1 : C P U
- 2 : チップセット
- 3 : R A M
- 4 : ハードディスクコントローラ
- 5 : ディスプレイコントローラ
- 6 : ハードディスクドライブ
- 7 : C D - R O M ドライブ
- 8 : ディスプレイ
- 11 : C P U とチップセットを繋ぐ信号線

12：チップセットとRAMを繋ぐ信号線

13：チップセットと各種周辺機器とを繋ぐ周辺機器バス

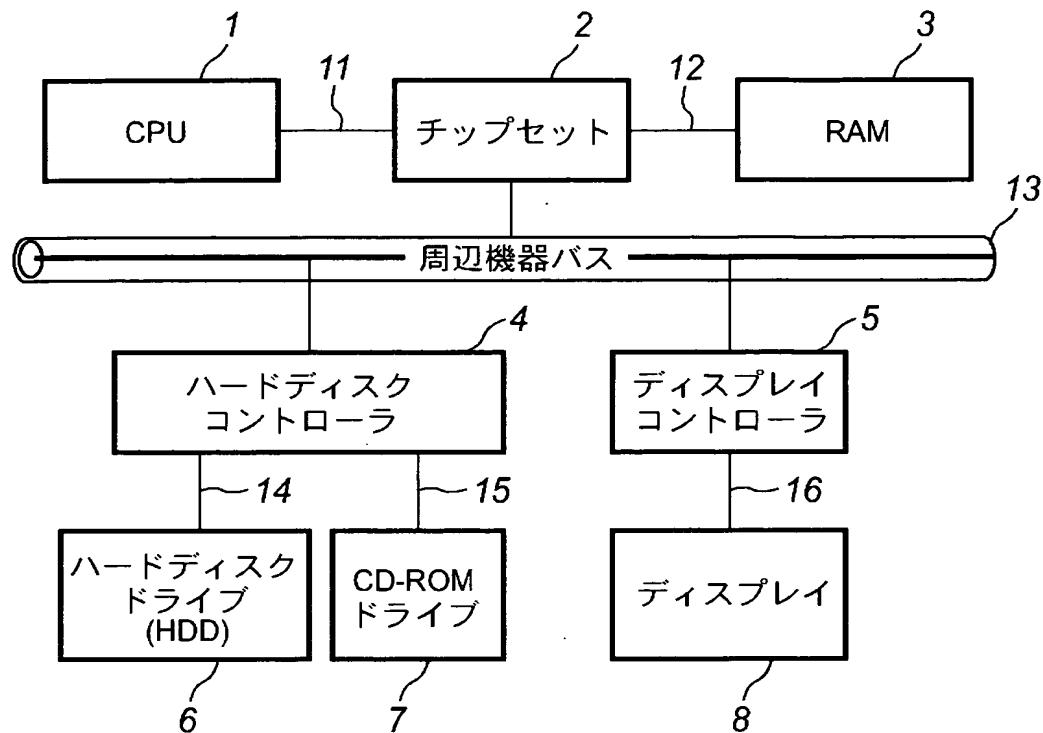
14：ハードディスクコントローラとハードディスクドライブを繋ぐ信号線

15：ハードディスクコントローラとCD-ROMドライブを繋ぐ信号線

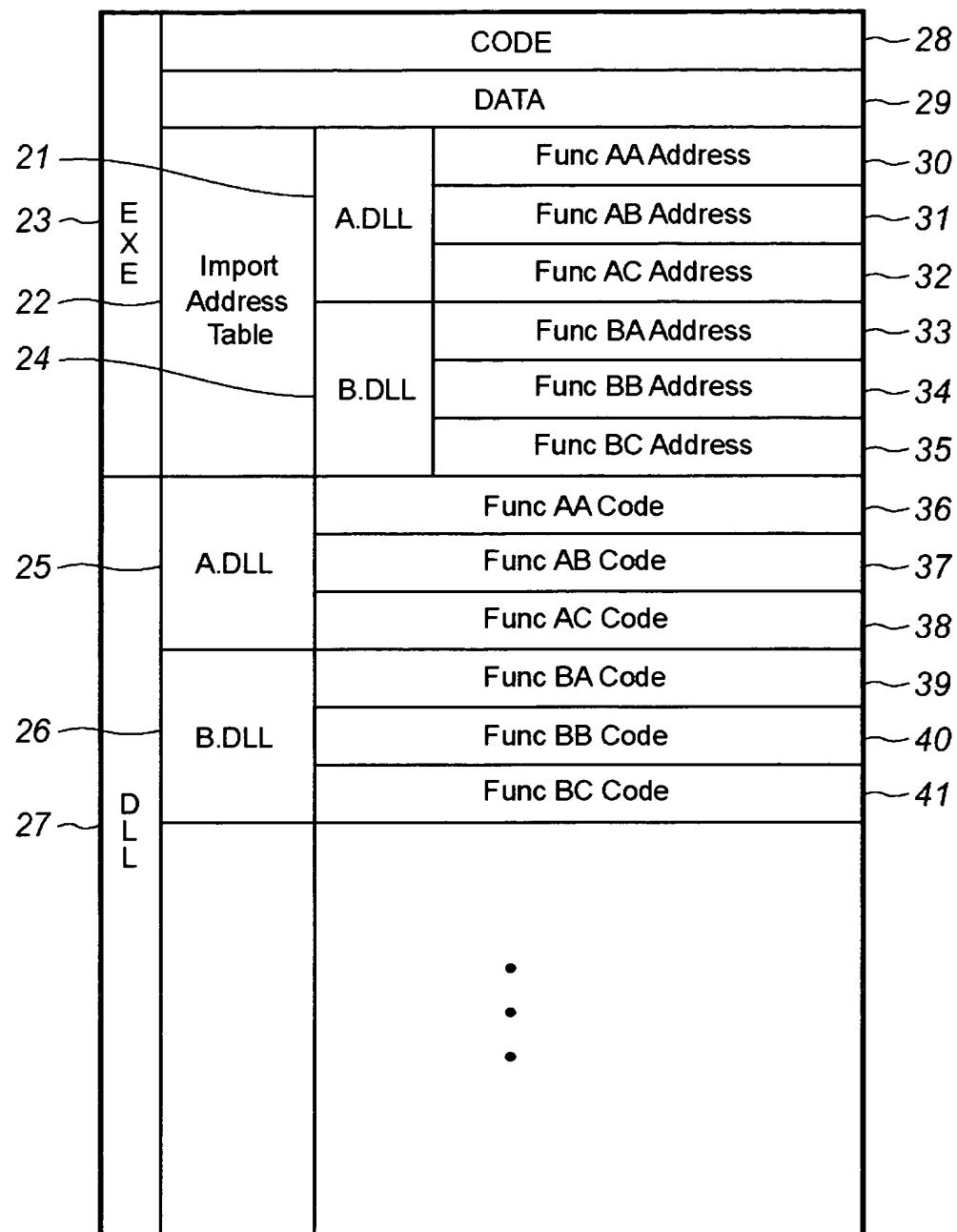
16：ディスプレイコントローラとディスプレイを繋ぐ信号線

【書類名】 図面

【図 1】



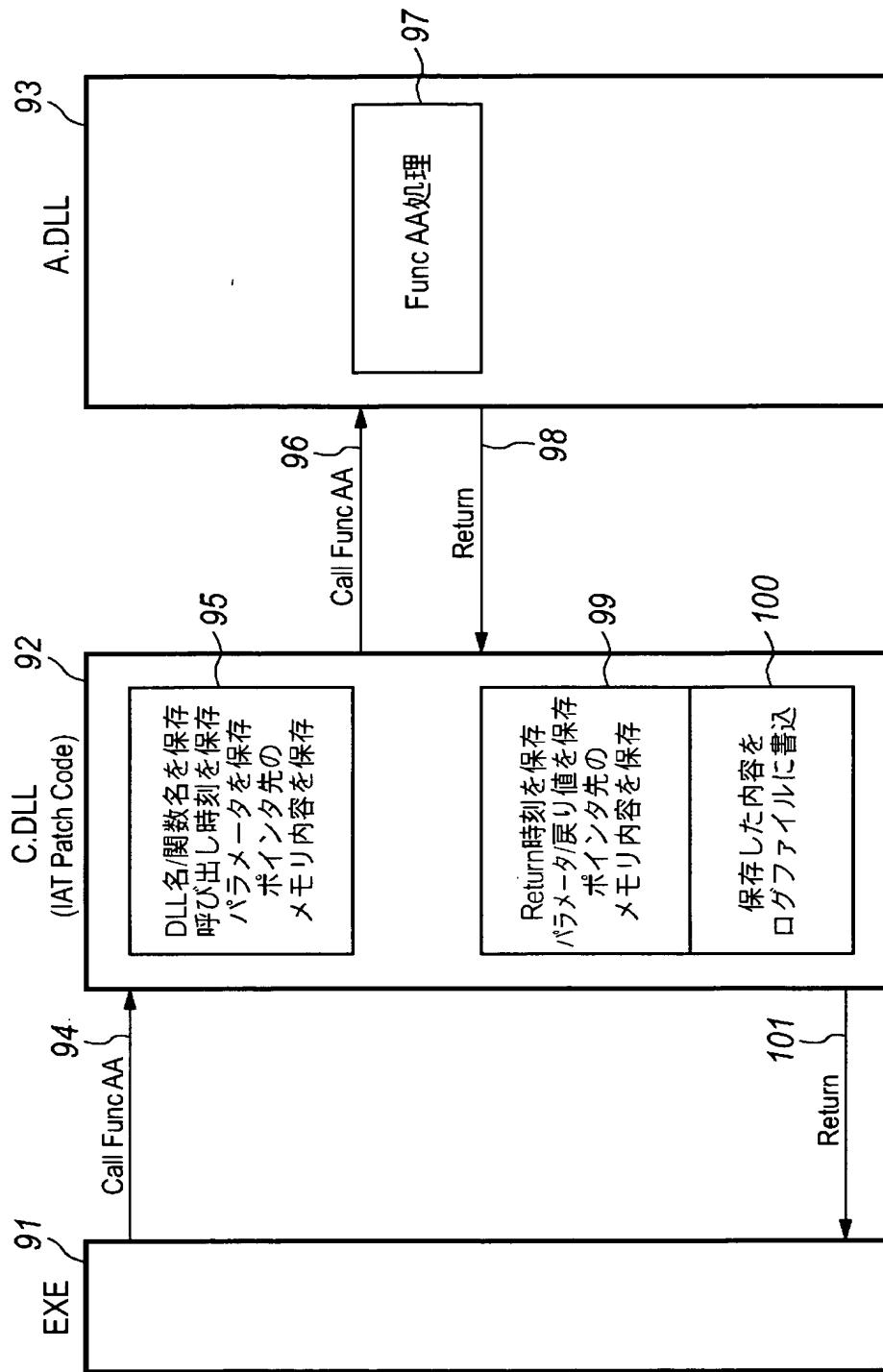
【図2】



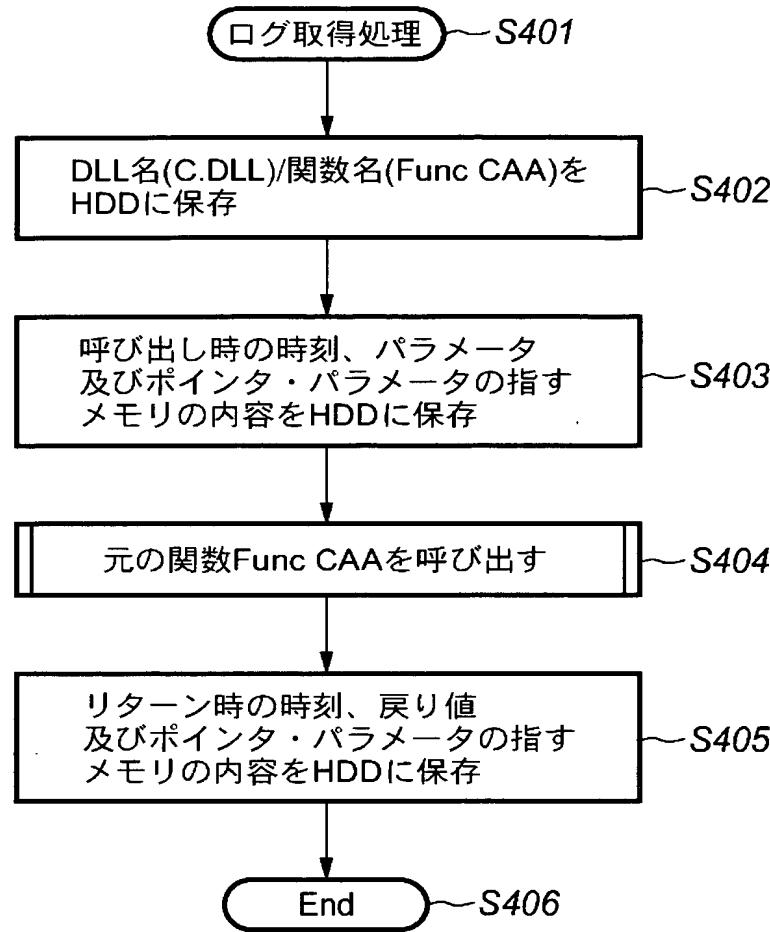
【図3】

	CODE		~59		
	DATA		~60		
51		Func CAA Address	~61		
53	E X E	Import Address Table	A.DLL	Func CAB Address	~62
52				Func CAC Address	~63
54			B.DLL	Func CBA Address	~64
				Func CBB Address	~65
				Func CBC Address	~66
55			A.DLL	Func AA Code	~67
				Func AB Code	~68
				Func AC Code	~69
56	D L L	B.DLL		Func BA Code	~70
				Func BB Code	~71
				Func BC Code	~72
57		C.DLL		Func CAA Code (Call Fcnk AA)	~73
				Func CAB Code (Call Fcnk AB)	~74
				Func CAC Code (Call Fcnk AC)	~75
58				Func CBA Code (Call Fcnk BA)	~76
				Func CBB Code (Call Fcnk BB)	~77
				Func CBC Code (Call Fcnk BC)	~78
				：	

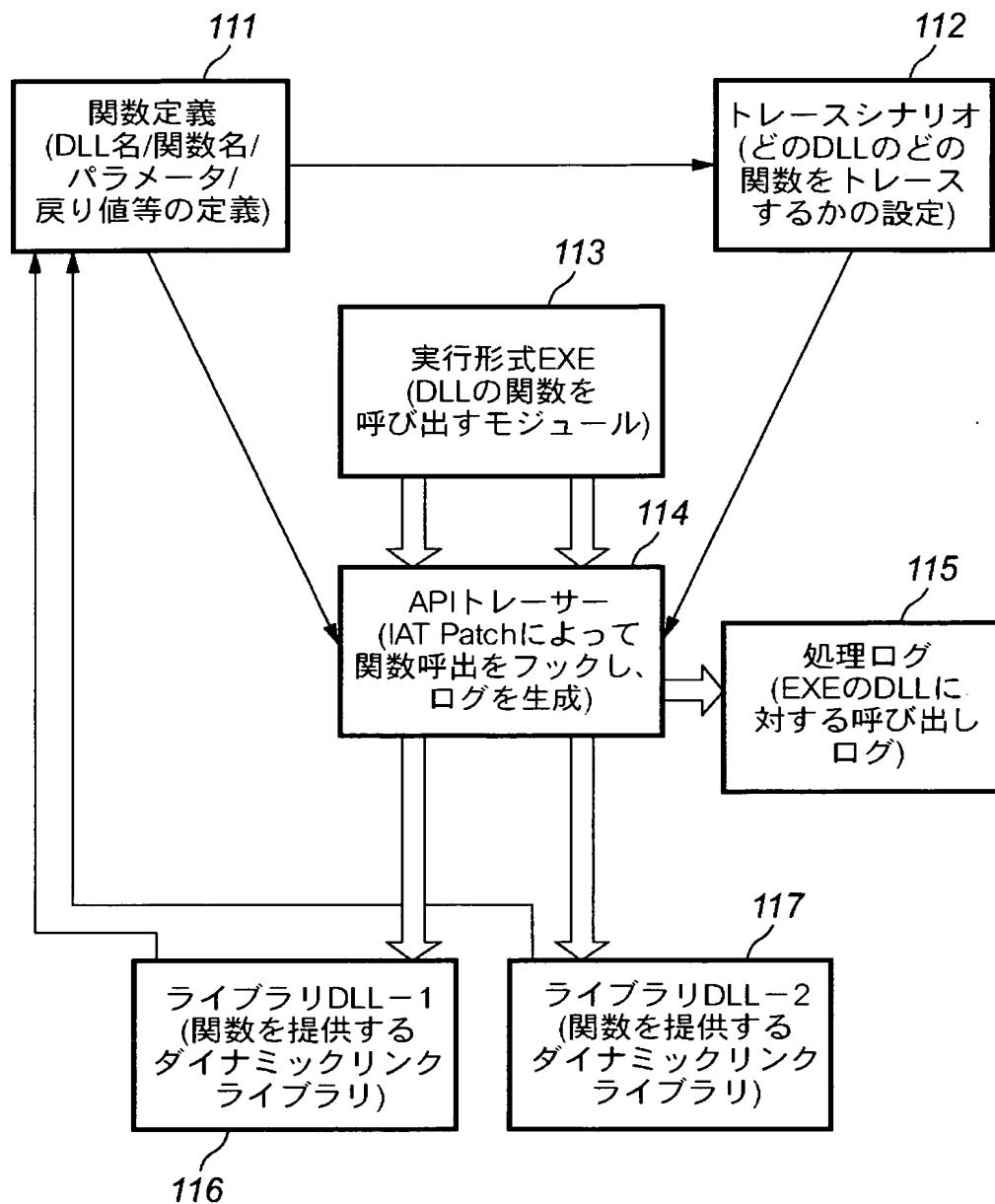
【図4A】



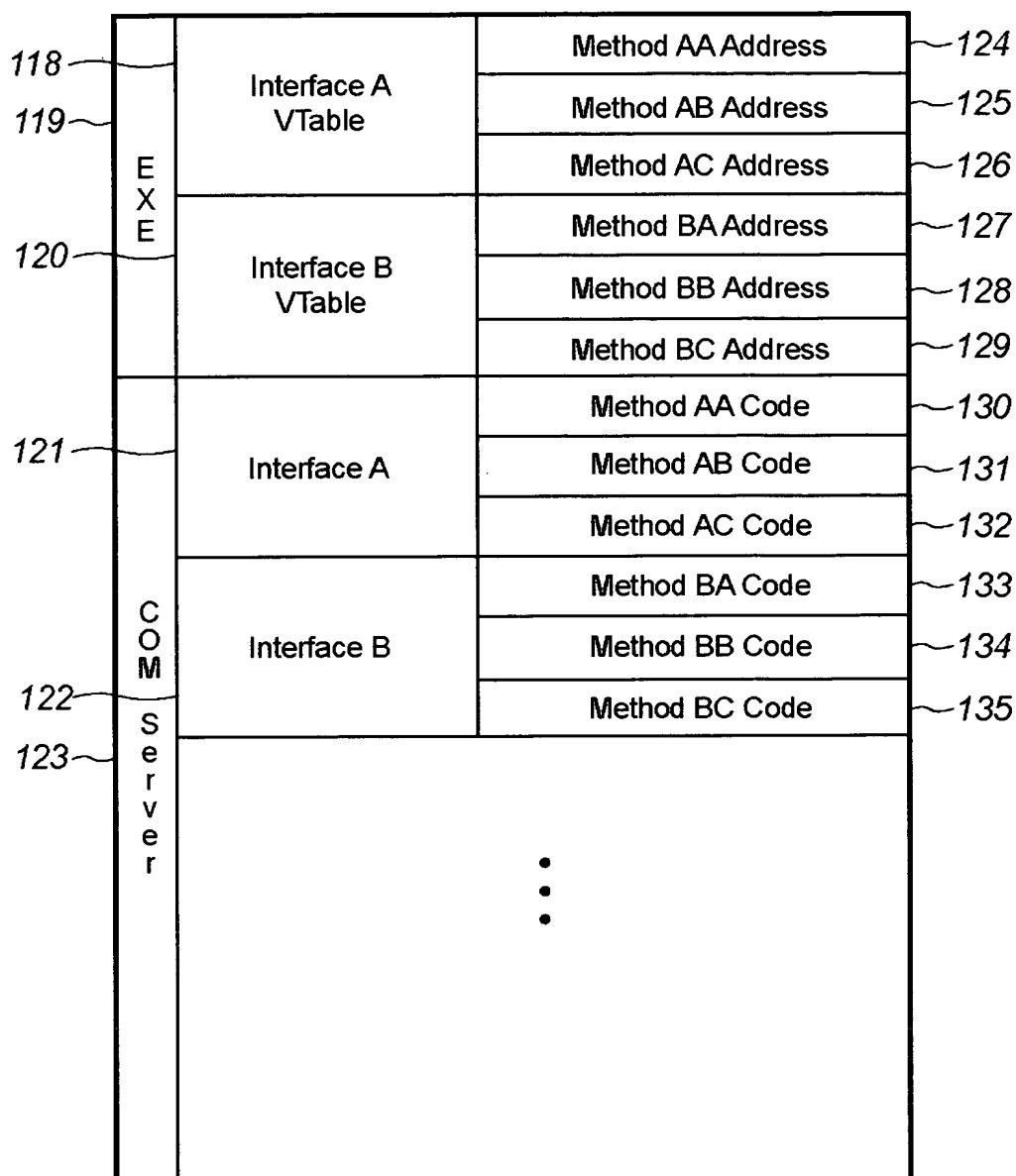
【図4B】



【図5】



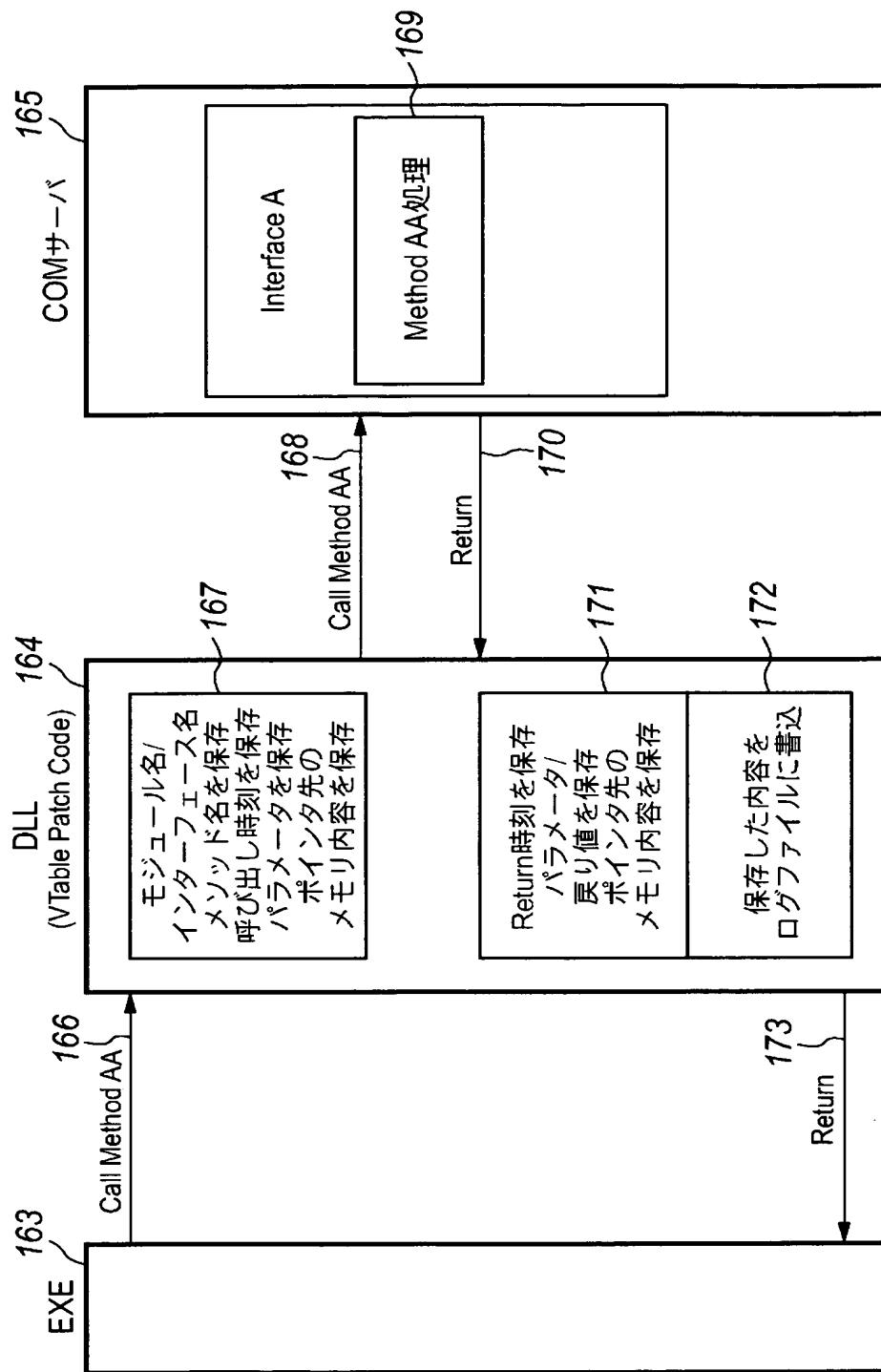
【図 6】



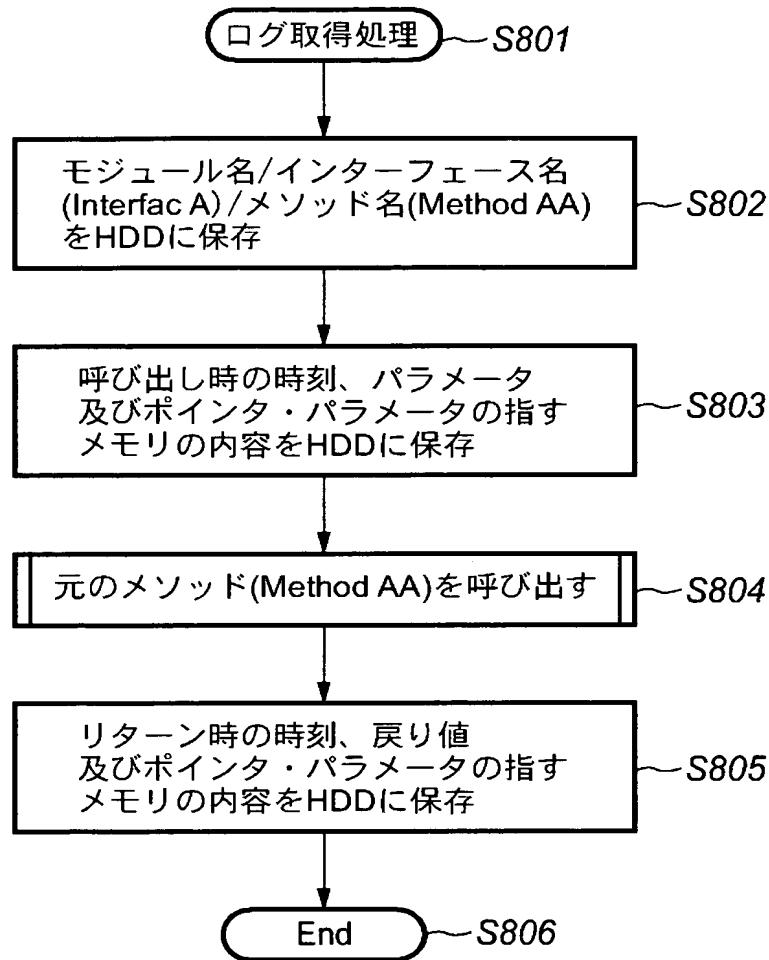
【図 7】

136	Interface A' VTable	Method A' A Address	～145
137		Method A' B Address	～146
EX E		Method A' C Address	～147
138	Interface B' VTable	Method B' A Address	～148
		Method B' B Address	～149
		Method B' C Address	～150
139	Interface A	Method AA Code	～151
COM		Method AB Code	～152
		Method AC Code	～153
141	Interface B	Method BA Code	～154
Server		Method BB Code	～155
		Method BC Code	～156
140	• •		
142	Interface A'	Method A' A Code (Call Method AA)	～157
DLL		Method A' B Code (Call Method AB)	～158
		Method A' C Code (Call Method AC)	～159
143	Interface B'	Method B' A Code (Call Method BA)	～160
		Method B' B Code (Call Method BB)	～161
144		Method B' C Code (Call Method BC)	～162

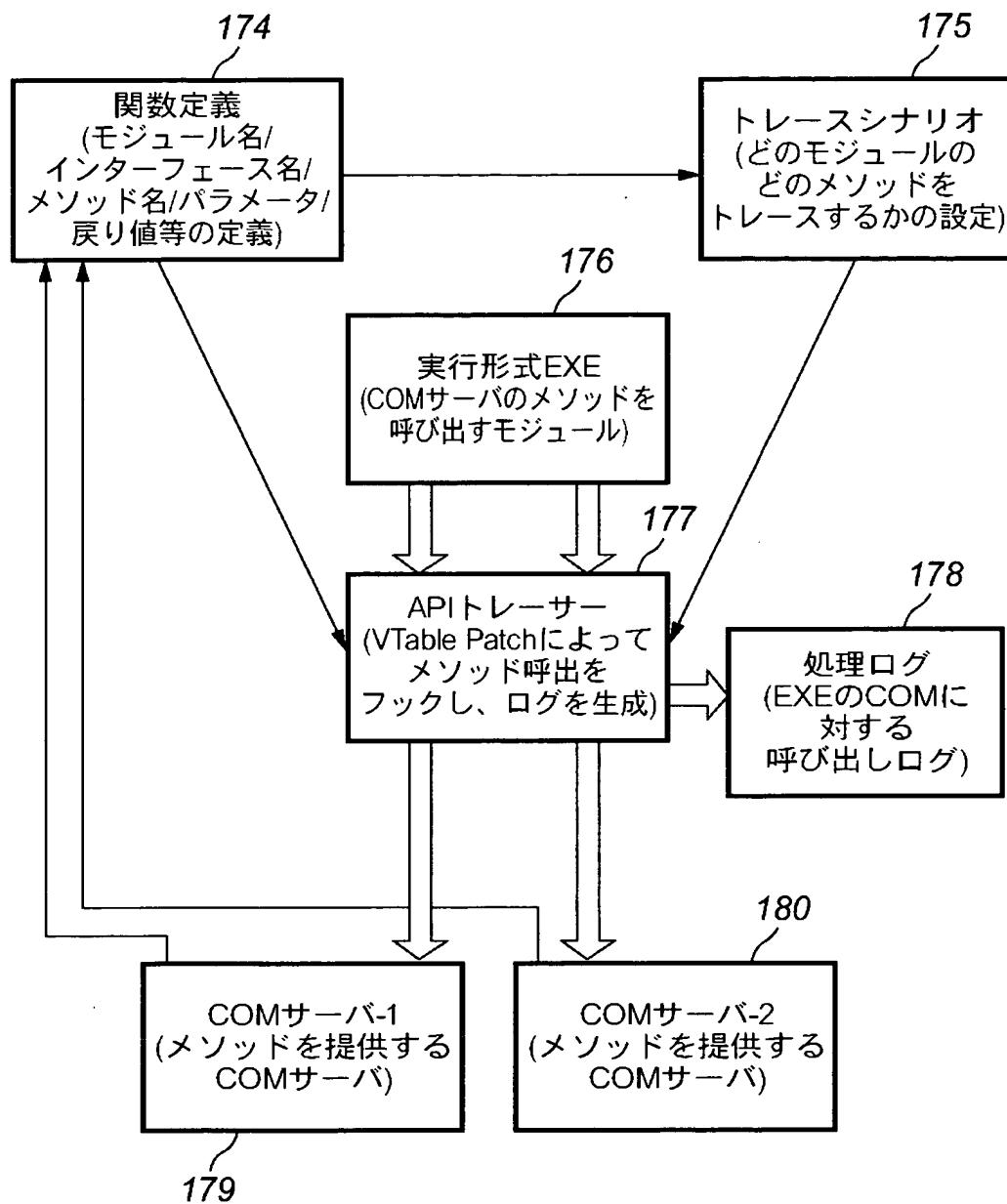
【図8A】



【図 8 B】



【図9】



## 【図10】

Interface名： Interface A  
Method名： Method AA  
引数： DWORD dwID  
戻り値： DWORD dwRet

Interface名： Interface A  
Method名： Method AB  
引数： DWORD dwSize  
戻り値： int nRet

Interface名： Interface A  
Method名： Method AC  
引数： DWORD dwSize  
戻り値： DWORD dwRet

Interface名： Interface B  
Method名： Method BA  
引数： DWORD dwSize  
戻り値： int nRet

Interface名： Interface B  
Method名： Method BC  
引数： DWORD dwID  
戻り値： DWORD dwRet

## 【図11】

Interface名 :	Interface A
Method名 :	Method AA
引数 :	DWORD dwID : 256
戻り値 :	DWORD dwRet : 0
In時刻 :	2002/03/25 22:24:12.025
Out時刻 :	2002/03/25 22:24:12.035

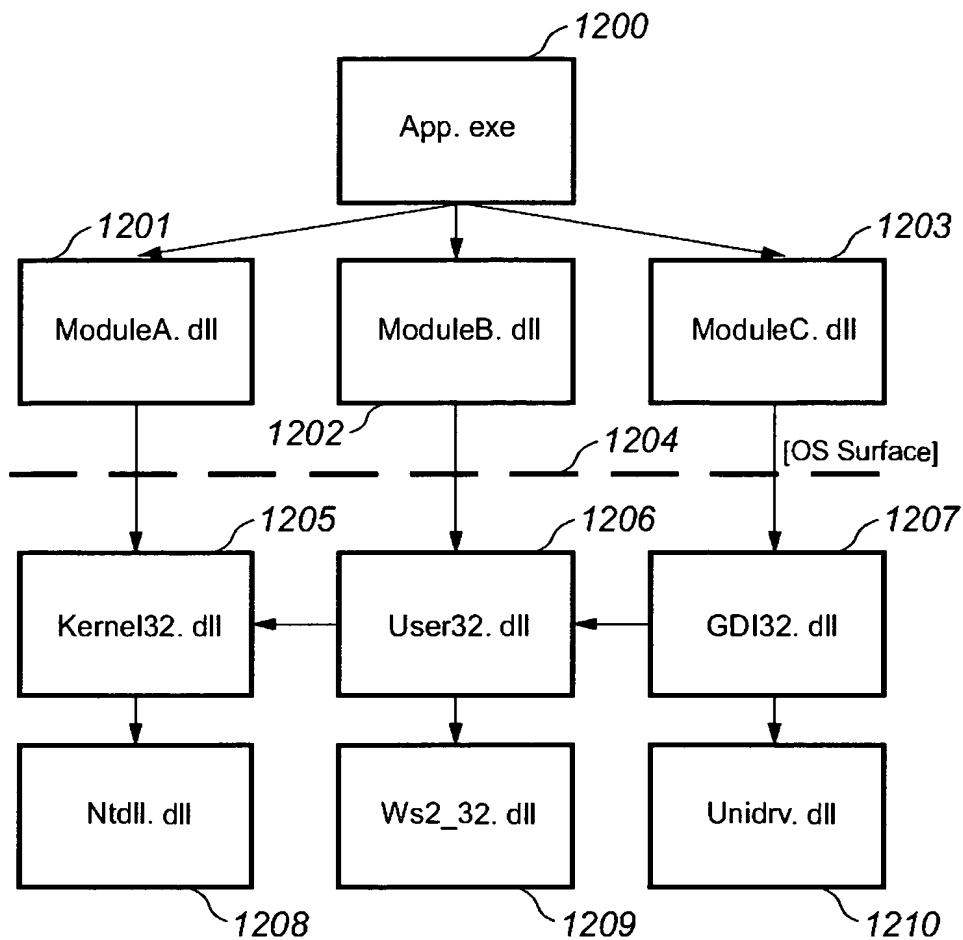
Interface名 :	Interface B
Method名 :	Method BA
引数 :	DWORD dwSize : 512
戻り値 :	int nRet : -1
In時刻 :	2002/03/25 22:24:12.046
Out時刻 :	2002/03/25 22:24:12.057

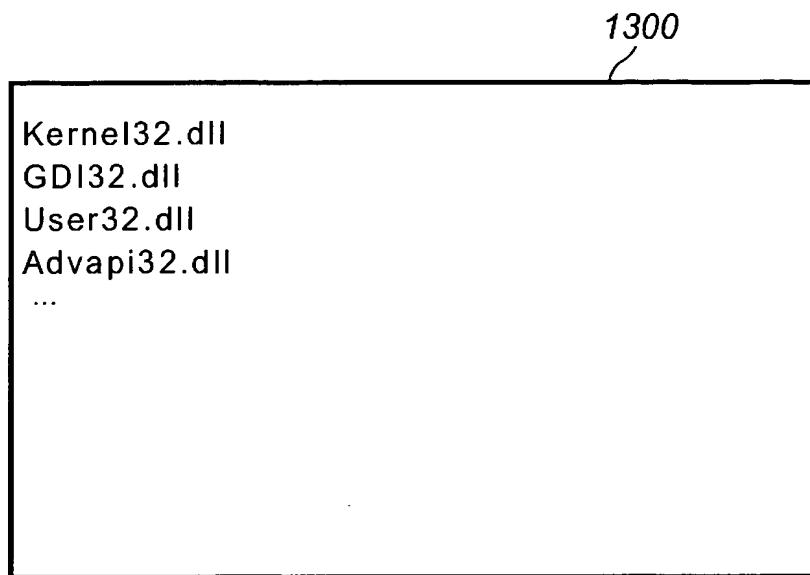
Interface名 :	Interface B
Method名 :	Method BC
引数 :	DWORD dwID : 12
戻り値 :	DWORD dwRet : 2
In時刻 :	2002/03/25 22:24:12.068
Out時刻 :	2002/03/25 22:24:12.079

...

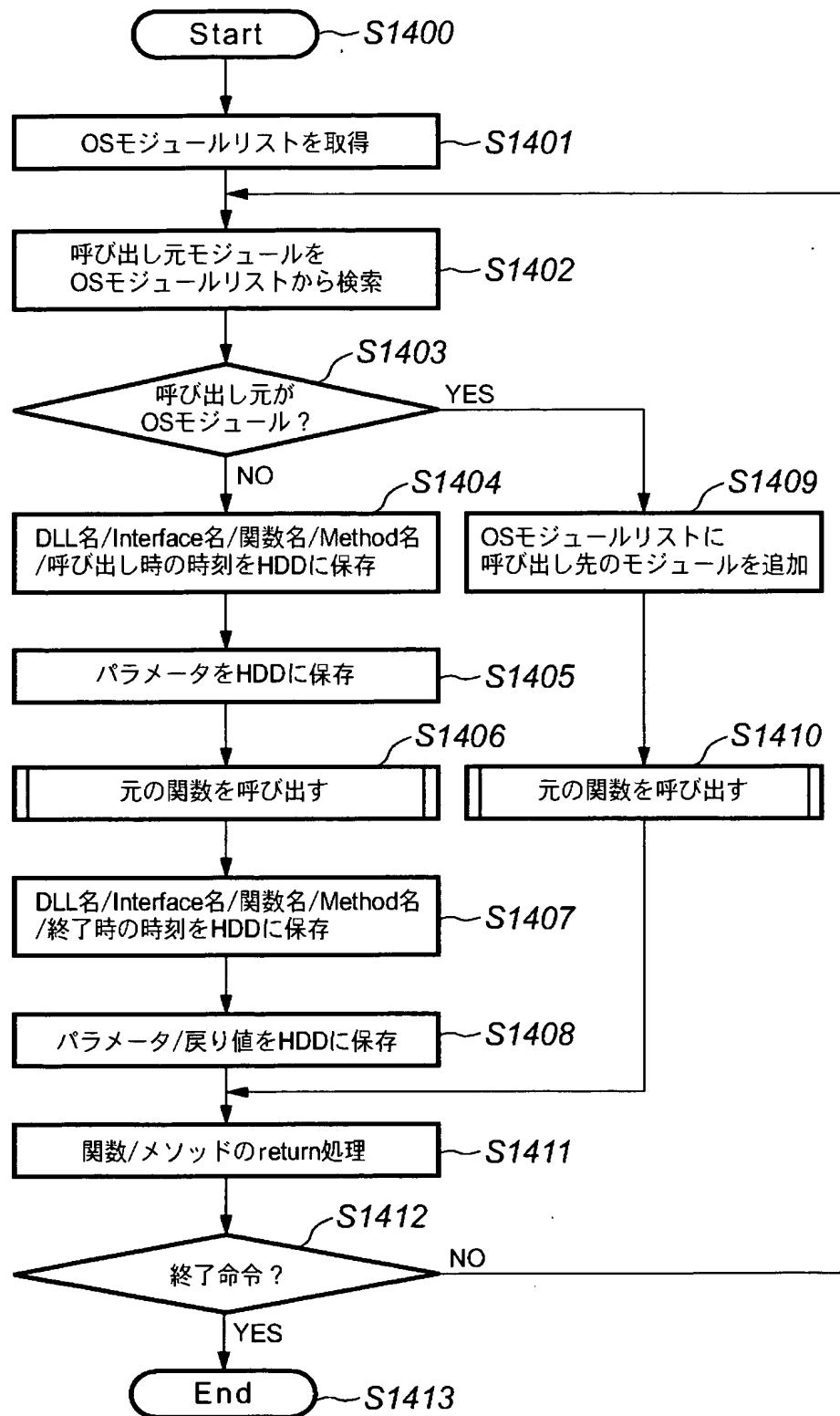
【図12】



【図13】



【図14】



## 【図 15 A】

1500

モジュール名 ModuleA.dll  
関数名 FuncA  
In引数 int nIndex : 3  
Out引数 DWORD \* nRet : 0x007a61c0/0x0000000a(10)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:12.025  
Out時刻 2002/03/25 22:24:12.035

モジュール名 Kernel32.dll  
関数名 FuncB  
In引数 int nIndex : 1  
Out引数 DWORD \* nRet : 0x007a625c/0x0000000b(11)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:12.046  
Out時刻 2002/03/25 22:24:12.057

モジュール名 ModuleB.dll  
関数名 FuncD  
In引数 int nIndex : 4  
Out引数 DWORD \* nRet : 0x007b61c0/0x0000000d(13)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:12.089  
Out時刻 2002/03/25 22:24:13.000

モジュール名 User32.dll  
関数名 FuncE  
In引数 int nIndex : 6  
Out引数 DWORD \* nRet : 0x002a61c0/0x0000000e(14)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:13.011  
Out時刻 2002/03/25 22:24:13.022

...

## 【図 15 B】

1300

モジュール名 ModuleA.dll  
関数名 FuncA  
In引数 int nIndex : 3  
Out引数 DWORD\* nRet : 0x007a61c0/0x0000000a(10)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:12.025  
Out時刻 2002/03/25 22:24:12.035

モジュール名 Kernel32.dll  
関数名 FuncB  
In引数 int nIndex : 1  
Out引数 DWORD\* nRet : 0x007a625c/0x0000000b(11)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:12.046  
Out時刻 2002/03/25 22:24:12.057

モジュール名 Ntdll.dll  
関数名 FuncC  
In引数 int nIndex : 2  
Out引数 DWORD\* nRet : 0x007a6122/0x0000000c(12)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:12.068  
Out時刻 2002/03/25 22:24:12.079

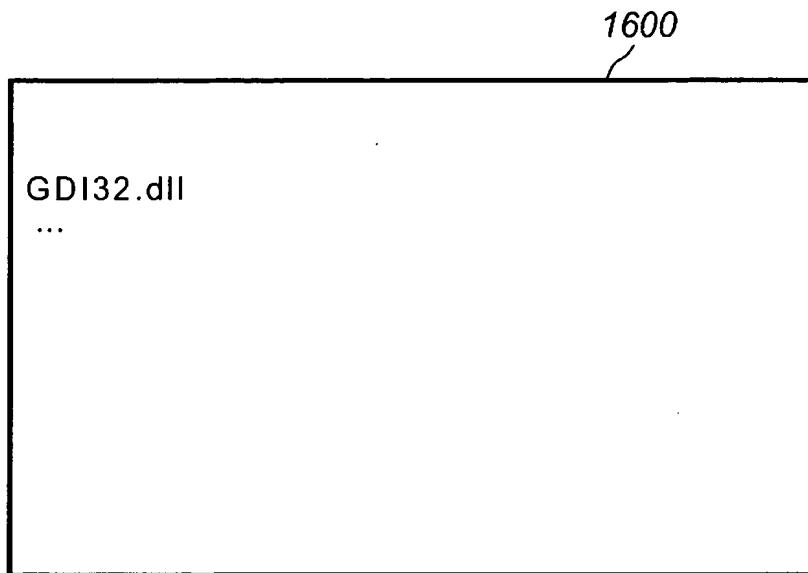
モジュール名 ModuleB.dll  
関数名 FuncD  
In引数 int nIndex : 4  
Out引数 DWORD\* nRet : 0x007b61c0/0x0000000d(13)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:12.089  
Out時刻 2002/03/25 22:24:13.000

モジュール名 User32.dll  
関数名 FuncE  
In引数 int nIndex : 6  
Out引数 DWORD\* nRet : 0x002a61c0/0x0000000e(14)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:13.011  
Out時刻 2002/03/25 22:24:13.022

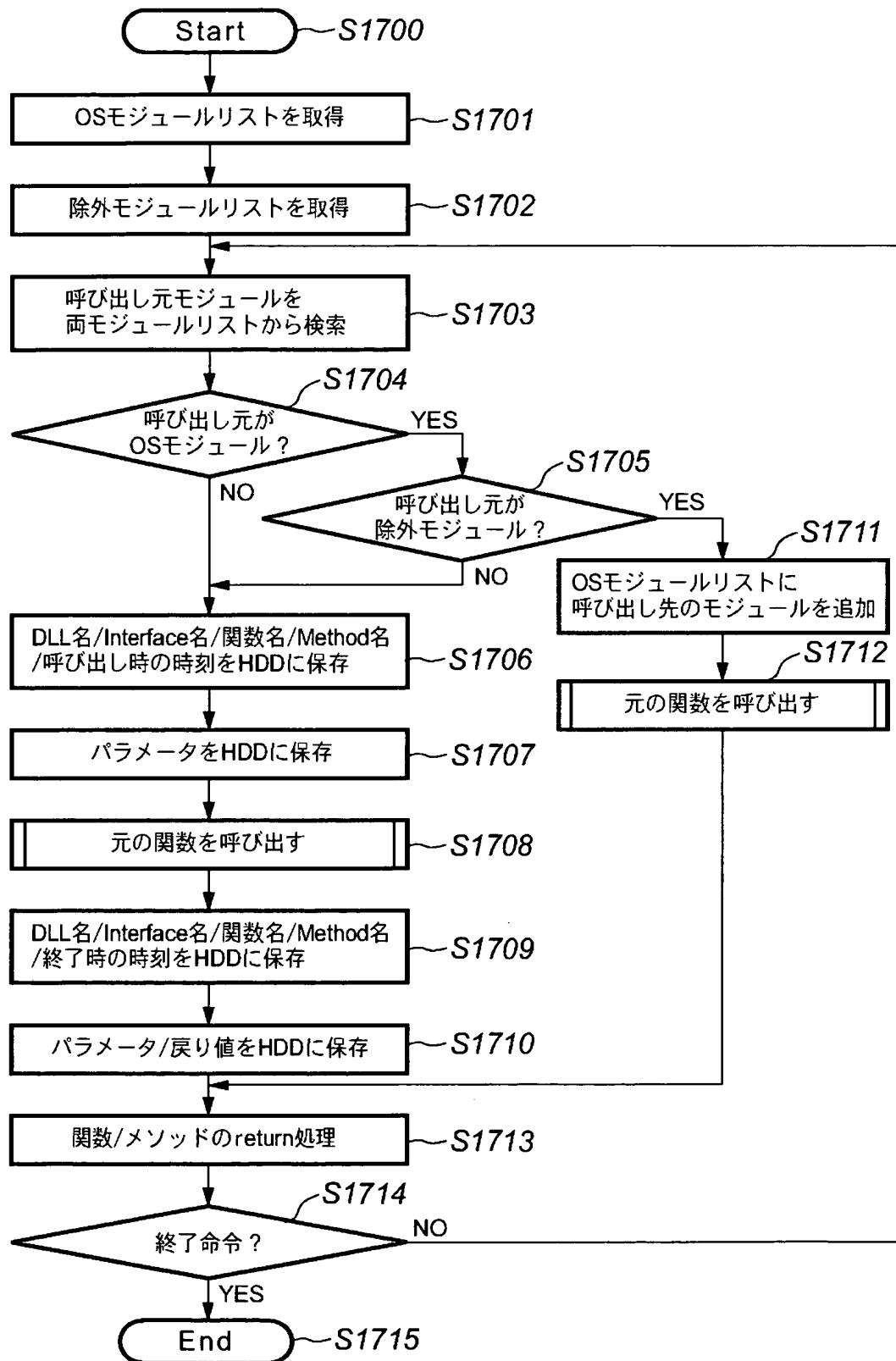
モジュール名 Ws2\_32.dll  
関数名 FuncF  
In引数 int nIndex : 5  
Out引数 DWORD\* nRet : 0x002a6dc0/0x0000000f(15)  
戻り値 int : 0  
In時刻 2002/03/25 22:24:13.034  
Out時刻 2002/03/25 22:24:13.055

...

【図 16】



【図17】



## 【図 18】

```
[  
    uuid(58DB5633-0694-4340-97CE-4E1AC6BFFBA7),      /TestDllStd.  
    helpstring("TestDllStd Type Library For PAT"),  
    version(1.0)  
}  
  
library TestDllStd  
{  
    typedef [public] struct  
    {  
        char chParam;  
        unsigned char uchParam;  
        short sParam;  
        unsigned short usParam;  
        int nParam;  
        unsigned int unParam;  
        long lParam;  
        unsigned long ulParam;  
        double dbParam;  
        float fParam;  
    } TESTSTRUCT;  
    typedef [public] TESTSTRUCT *LPTESTSTRUCT;  
  
    //DEFINE_GUID(GUID_PROGID, 0x8e037d65, 0xefa0, 0x40e7, 0x91, 0x43, 0xef, 0x70, 0x56, 0x94, 0x5b,  
    0x79);  
    [  
        uuid(8E037D65-EFA0-40e7-9143-EF7056945B79),  
        helpstring("TestDllStd.dll for PAT object."),  
    ]  
    interface  
    test  
    {  
        char _stdcall FuncCharStd([in] char chParam);  
        char* _stdcall FuncPCharStd([in, out] char* lpchParam);  
  
        TESTSTRUCT _stdcall FuncStructStd([in]TESTSTRUCT TestStruct);  
        LPTESTSTRUCT _stdcall FuncPStructStd([in, out]LPTESTSTRUCT lpTestStruct);  
    };  
}
```

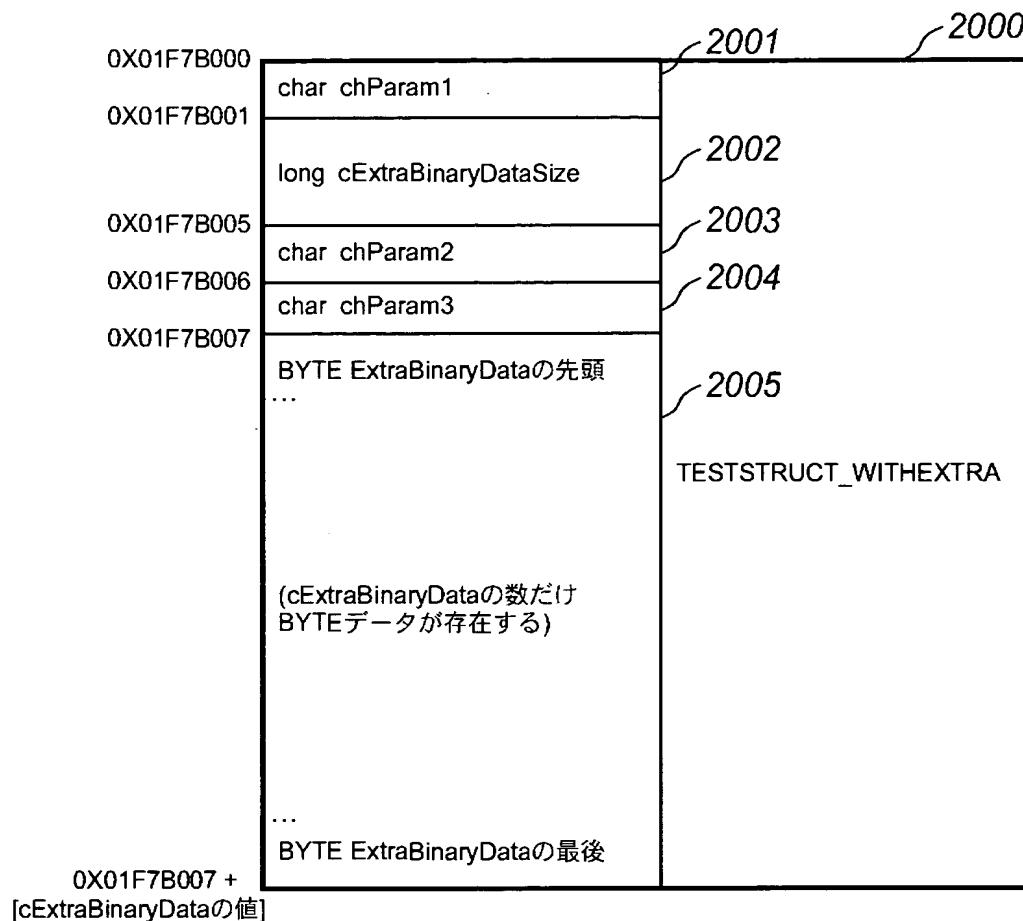
【図 19】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000 1900

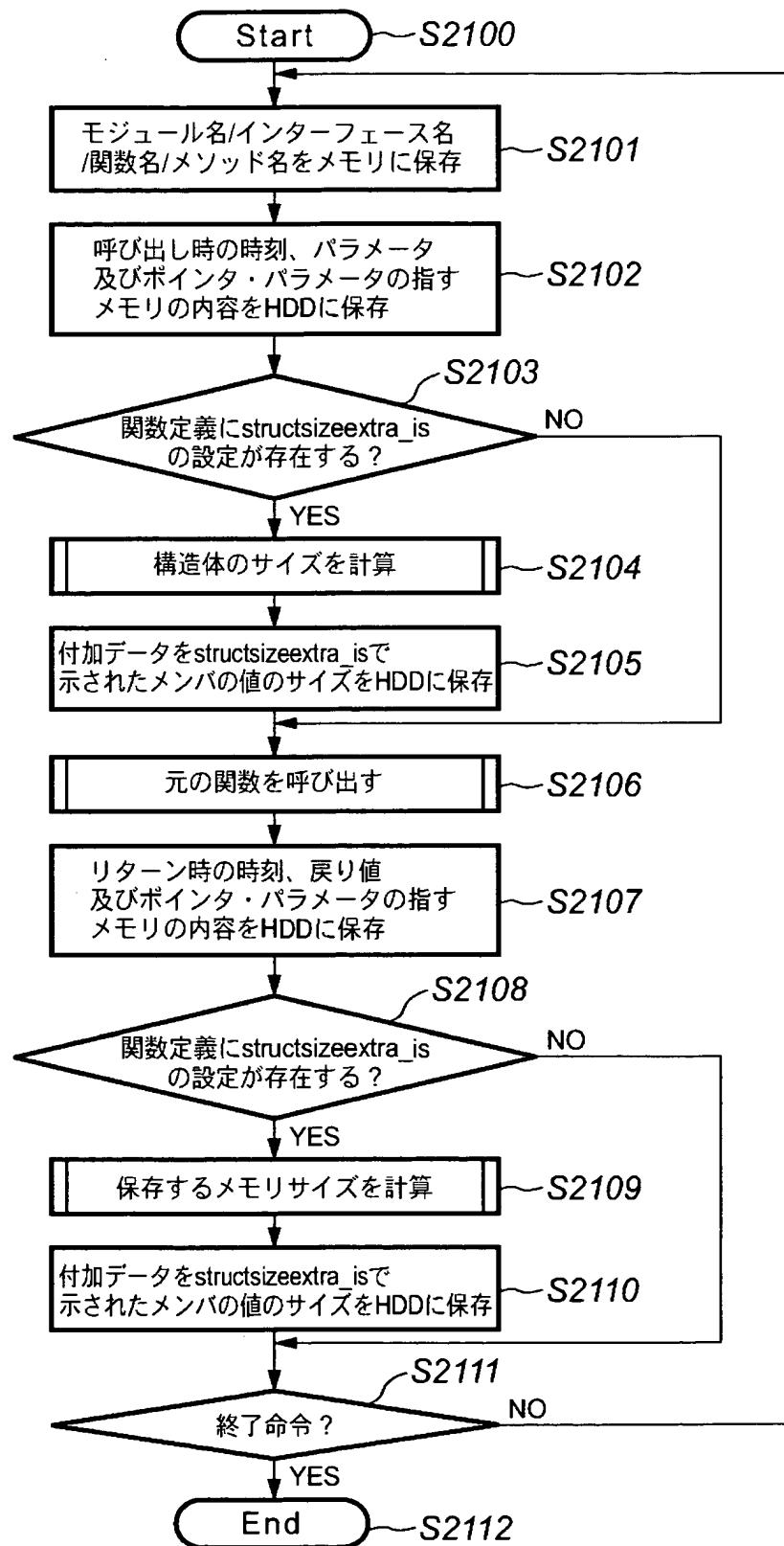
typedef [public] struct
{
    char chParam1;
    [custom(PAT_PARAM_ATTR_ID, "structsizeextra_is()")] long cExtraBinaryDataSize; 1901
    char chParam2;
    char chParam3;
} TESTSTRUCT_WITHEXTRA;

interface
test
{
    void FuncStructsizeextras 1902
    (
        [out] TESTSTRUCT_WITHEXTRA* lpParam
    );
}
```

【図20】



【図 21】



【図 22】

		2200	2201
モジュール名：	TestDllStd.DLL		
関数名：	Func Structsizeextras		
引数(in)：			
引数(out)：	struct TESTSTRUCT_WITHEXTRA: char chParam1 : "a" long cExtraBinaryDataSize: 8 char chParam2 : "b" char chParam3 : "c" ExtraBinaryData : 0x01F7B007. Data ID=0x0001		
戻り値：	void :		
In時刻：	2002/03/25 22:24:12.025		
Out時刻：	2002/03/25 22:24:12.035		
モジュール名：	TestDllStd.DLL		
関数名：	Func Structsizeextras		
引数(in)：			
引数(out)：	struct TESTSTRUCT_WITHEXTRA: char chParam1 : "d" long cExtraBinaryDataSize: 40 char chParam2 : "e" char chParam3 : "f" ExtraBinaryData : 0x01F7B007. Data ID=0x0002		
戻り値：	void :		
In時刻：	2002/03/25 22:24:12.046		
Out時刻：	2002/03/25 22:24:12.057		
モジュール名：	TestDllStd.DLL		
関数名：	Func Structsizeextras		
引数(in)：			
引数(out)：	struct TESTSTRUCT_WITHEXTRA: char chParam1 : "g" long cExtraBinaryDataSize: 5 char chParam2 : "h" char chParam3 : "i" ExtraBinaryData : 0x01F7B007. Data ID=0x0003		
戻り値：	void :		
In時刻：	2002/03/25 22:24:12.068		
Out時刻：	2002/03/25 22:24:12.079		
モジュール名：	TestDllStd.DLL		
関数名：	Func Structsizeextras		
引数(in)：			
引数(out)：	struct TESTSTRUCT_WITHEXTRA: char chParam1 : "j" long cExtraBinaryDataSize: 7 char chParam2 : "k" char chParam3 : "l" ExtraBinaryData : 0x01F7B007. Data ID=0x0004		
戻り値：	void :		
In時刻：	2002/03/25 22:24:12.100		
Out時刻：	2002/03/25 22:24:12.179		
...			
			Data ID: 0x0001 Size: 8 00000000: 10 00 00 00 4A 03 A5 20
			Data ID: 0x0002 Size: 40 00000000: 05 00 00 00 4A 03 A5 20 00000008: 05 00 00 00 4B 03 A5 20 00000010: 05 00 00 00 4C 03 A5 20 00000018: 05 00 00 00 4D 03 A5 20 00000020: 05 00 00 00 4E 03 A5 20
			Data ID: 0x0003 Size: 40 00000000: 66 4A 70 50 00
			Data ID: 0x0004 Size: 7 00000000: 01 5D 66 B2 20 49 20
			...

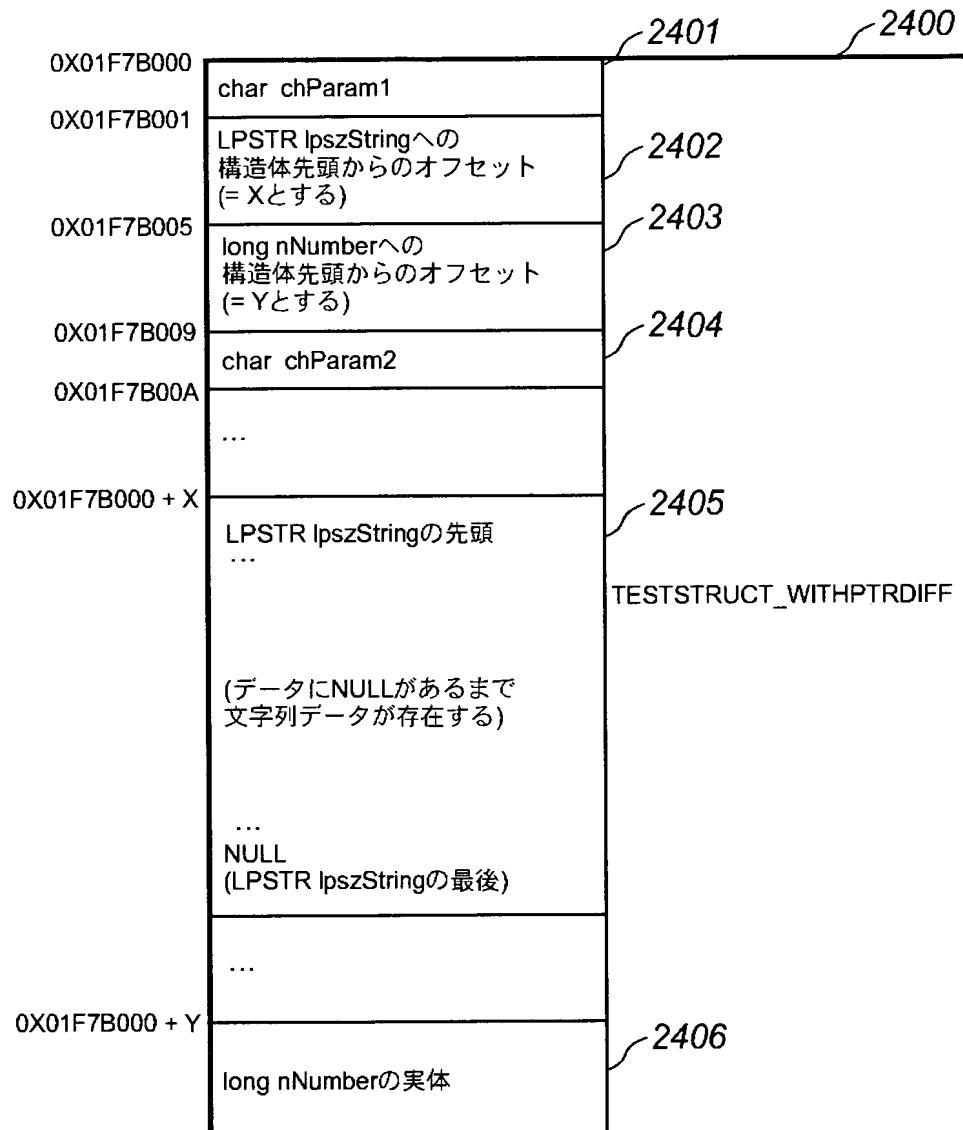
【図 23】

```
#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000 2300

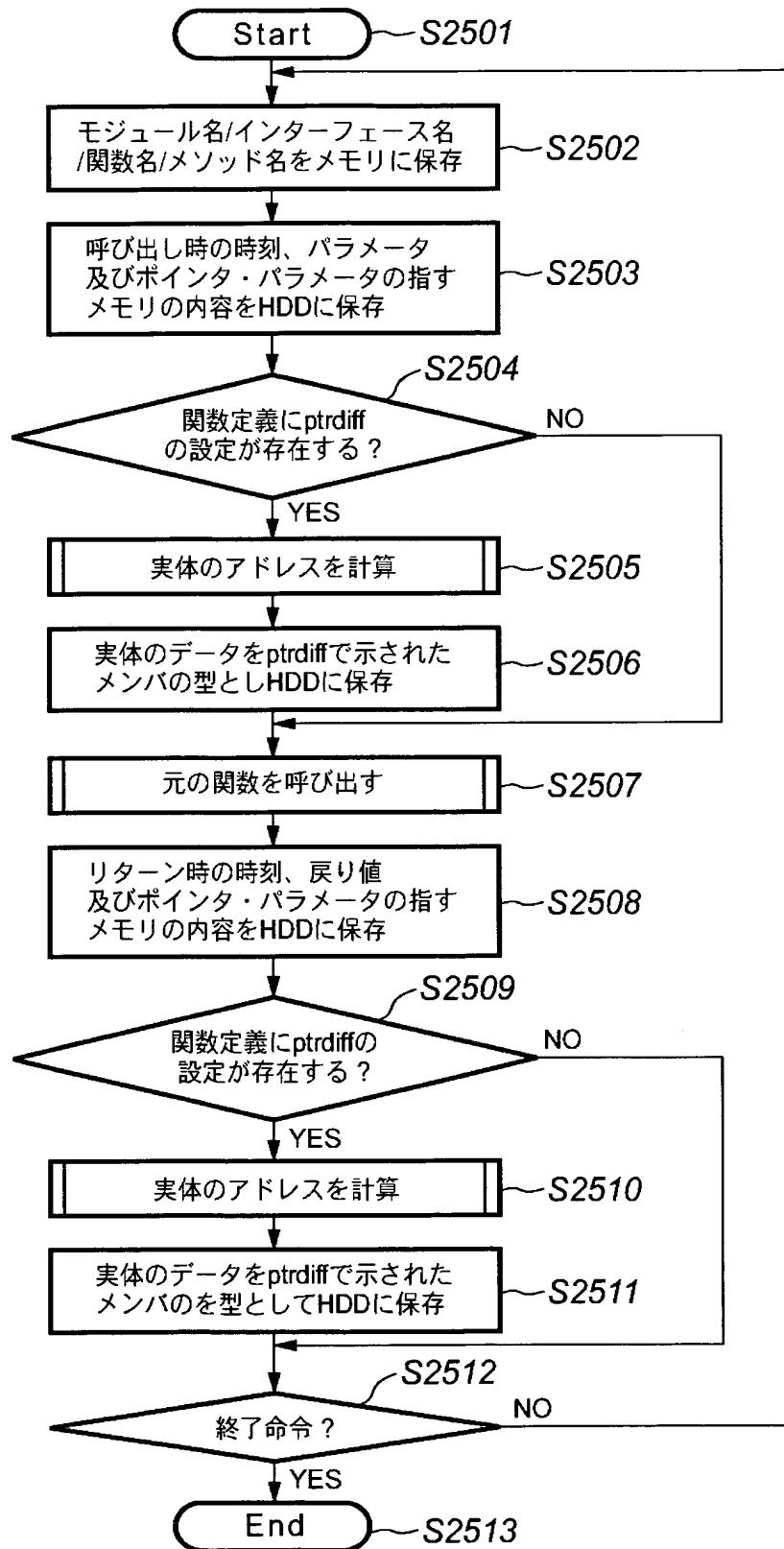
typedef [public] struct
{
    char chParam1; 2301
    [custom(PAT_PARAM_ATTR_ID, "ptrdiff()")] LPSTR lpszString;
    [custom(PAT_PARAM_ATTR_ID, "ptrdiff()")] long nNumber;
    char chParam2; 2302
} TESTSTRUCT_WITHPTRDIFF;

interface
test
{
    void FuncPtrdiff
    (
        [in] TESTSTRUCT_WITHPTRDIFF* lpParam 2303
    );
}
```

【図24】



【図25】



【図 26】

2600

```

モジュール名 : TestDllStd.DLL
関数名 : FuncPtdiff
引数(in) : struct TESTSTRUCT_WITHPTRDIFF {
    char chParam1 : "a"
    Offset to LPSTR IpszString : 16
    LPSTR IpszString : "Test1"
    Offset to long nNumber : 28
    long nNumber : 1
    char chParam3 : "b"
}
引数(out) :
戻り値 : void :
In時刻 : 2002/03/25 22:24:12.025
Out時刻 : 2002/03/25 22:24:12.035

モジュール名 : TestDllStd.DLL
関数名 : FuncPtdiff
引数(in) : struct TESTSTRUCT_WITHPTRDIFF {
    char chParam1 : "c"
    Offset to LPSTR IpszString : 18
    LPSTR IpszString : "Test2"
    Offset to long nNumber : 30
    long nNumber : 2
    char chParam3 : "d"
}
引数(out) :
戻り値 : void :
In時刻 : 2002/03/25 22:24:12.046
Out時刻 : 2002/03/25 22:24:12.057

モジュール名 : TestDllStd.DLL
関数名 : FuncPtdiff
引数(in) : struct TESTSTRUCT_WITHPTRDIFF {
    char chParam1 : "e"
    Offset to LPSTR IpszString : 20
    LPSTR IpszString : "Test3"
    Offset to long nNumber : 32
    long nNumber : 3
    char chParam3 : "f"
}
引数(out) :
戻り値 : void :
In時刻 : 2002/03/25 22:24:12.068
Out時刻 : 2002/03/25 22:24:12.079

モジュール名 : TestDllStd.DLL
関数名 : FuncPtdiff
引数(in) : struct TESTSTRUCT_WITHPTRDIFF {
    char chParam1 : "g"
    Offset to LPSTR IpszString : 16
    LPSTR IpszString : "Test4"
    Offset to long nNumber : 28
    long nNumber : 29
    char chParam3 : "h"
}
引数(out) :
戻り値 : void :
In時刻 : 2002/03/25 22:24:12.100
Out時刻 : 2002/03/25 22:24:12.179

```

【図 27】

```

#define PAT_PARAM_ATTR_ID 00000000-0000-0000-0000-000000000000 2700
interface test; 2701
typedef struct tagGUID {
    unsigned long Data1;
    unsigned short Data2;
    unsigned short Data3;
    unsigned char Data4[8];
} GUID;
interface test {
    HRESULT _stdcall DIIGetClassObject( 2702
        [in, custom(PAT_PARAM_ATTR_ID, "clsid_( )")] GUID* rctsid,
        [in] GUID* riid,
        [out, custom(PAT_PARAM_ATTR_ID, "iid_is(riid)")] void** ppv);
};

2703

```

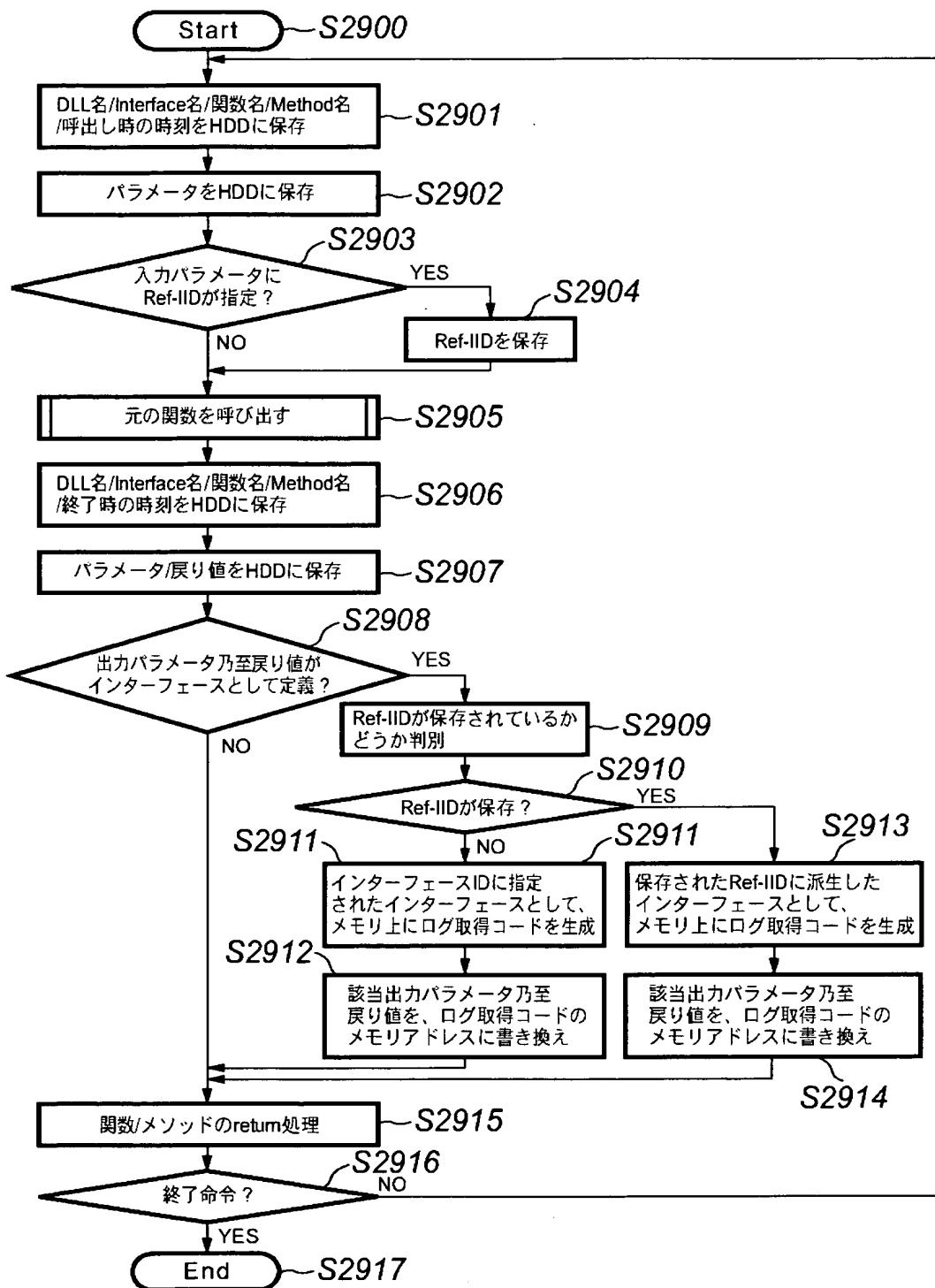
【図 28】

```

interface IGetInfo : IUnknown 2800
{
    HRESULT GetName(
        [in]     DWORD    dwID,
        [out]    LPSTR    lpszName
    );
    HRESULT FreeNameBuffer(
        [in]     LPSTR    lpszName
    );
}

```

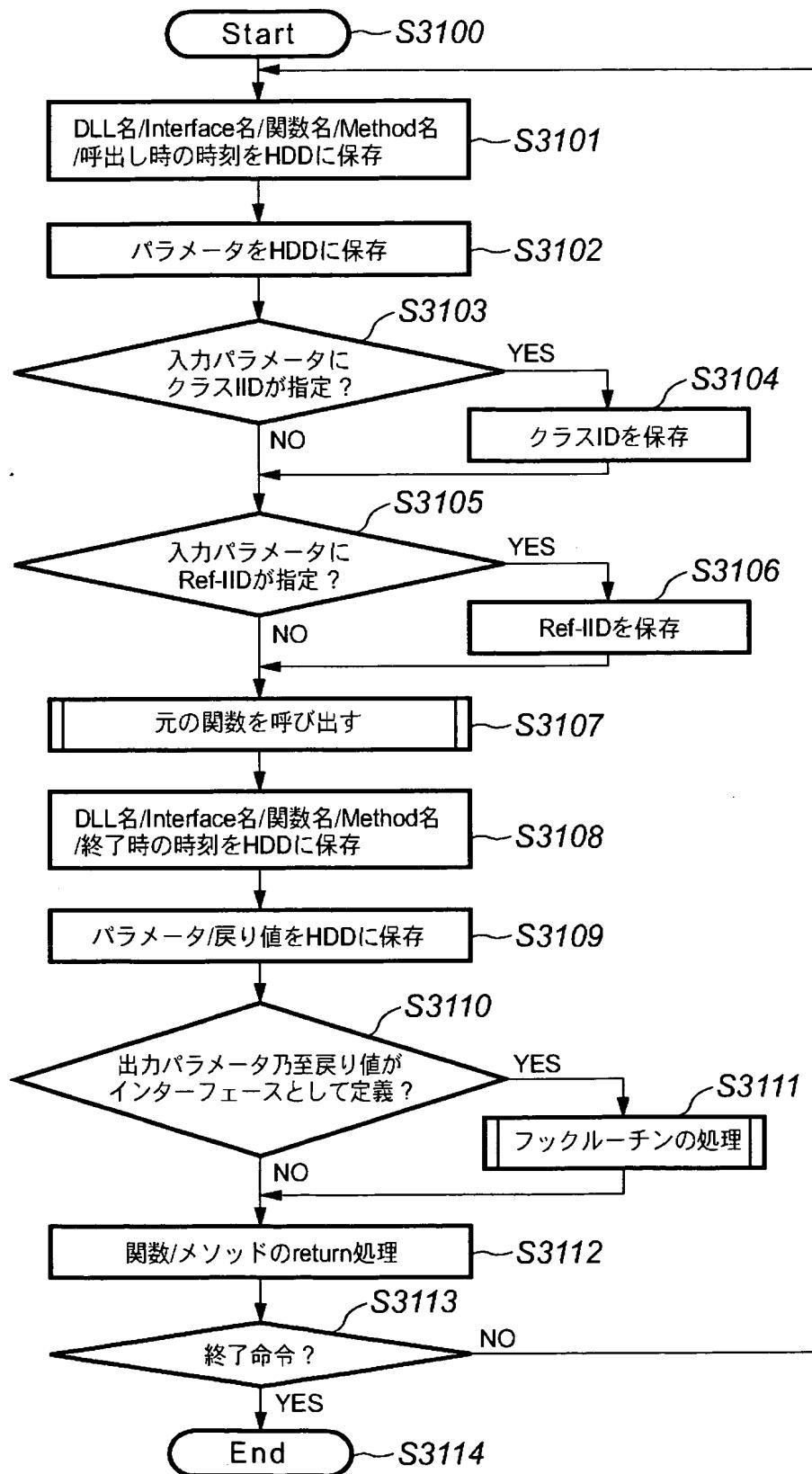
【図 29】



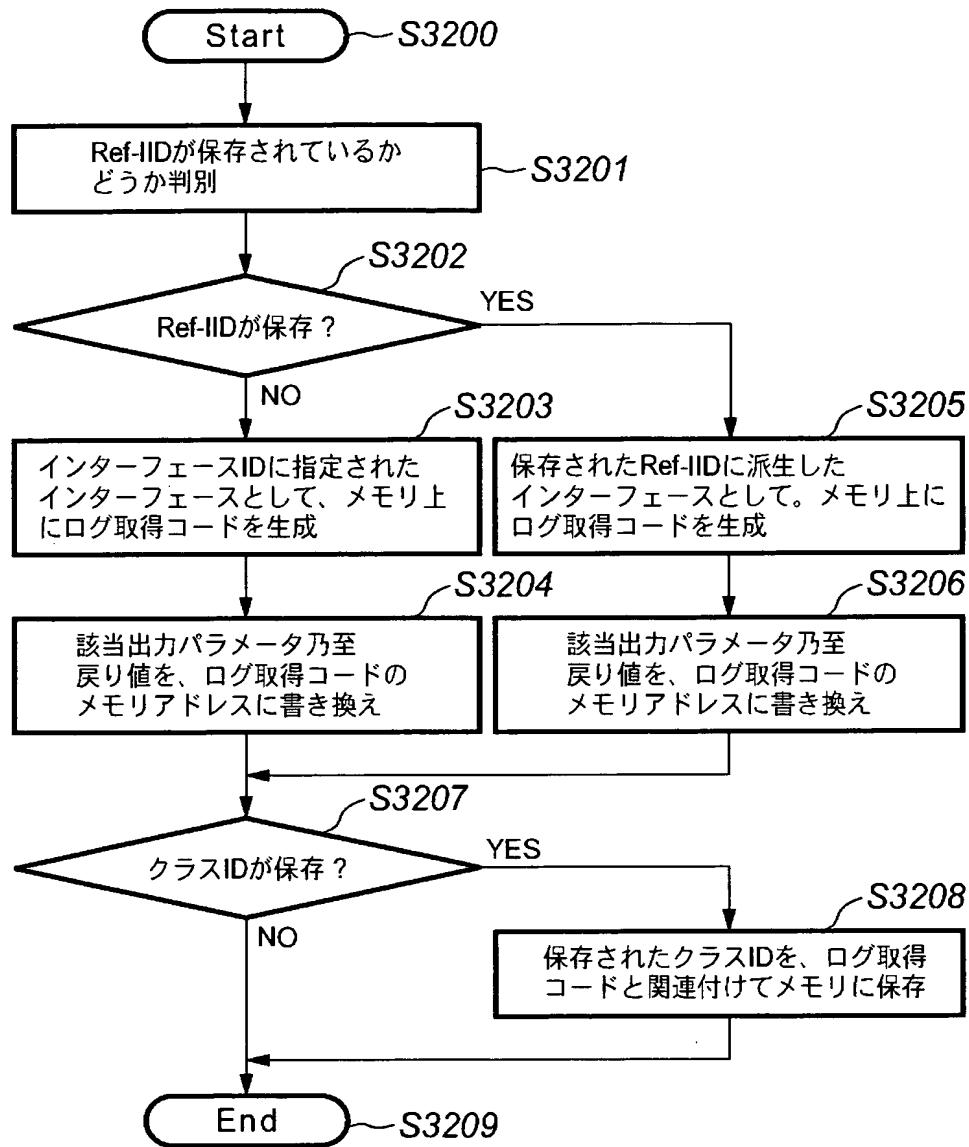
## 【図30】

モジュール名	GetInfo.dll	3000
関数名	DllGetClassObject	
In引数	GUID* rclsid : {abce80d7-9f46-11d1-882a-00c04fb961ec}	
	GUID* iid : {00000001-0000-0000-c000-0000000046}	
Out引数	void** ppv : 0x007a61c0/0x730e13e8(1930302440)	
戻り値	HRESULT : 0x00000000(0)	3001
In時刻	2002/03/25 22:24:12.025	
Out時刻	2002/03/25 22:24:12.035	3002
Interface名	GetInfo	3001
Method名	GetName	
In引数	DWORD dwID : 1	
	LPSTR lpszName : "Name-1"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:12.046	3002
Out時刻	2002/03/25 22:24:12.057	
Interface名	GetInfo	3002
Method名	FreeNameBuffer	
In引数	LPSTR lpszName : "Name-1"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:12.068	
Out時刻	2002/03/25 22:24:12.079	
Interface名	GetInfo	3002
Method名	GetName	
In引数	DWORD dwID : 2	
	LPSTR lpszName : "Name-2"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:12.089	3003
Out時刻	2002/03/25 22:24:13.000	
Interface名	GetInfo	3003
Method名	FreeNameBuffer	
In引数	LPSTR lpszName : "Name-2"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:13.011	
Out時刻	2002/03/25 22:24:13.022	
...		

【図 3 1】



【図32】



## 【図 3 3】

モジュール名	GetInfo.dll	3300
関数名	DIIGetClassObject	
In引数	GUID * rclsid : {abce80d7-9f46-11d1-882a-00c04fb961ec}	
	GUID * iid : {00000001-0000-0000-c000-0000000046}	
Out引数	void ** ppv : 0x007a61c0/0x730e13e8(1930302440)	
戻り値	HRESULT : 0x00000000(0)	3301
In時刻	2002/03/25 22:24:12.025	3302
Out時刻	2002/03/25 22:24:12.035	
モジュール名	GetInfo.dll	3303
Interface名	GetInfo	
Method名	GetName	
In引数	DWORD dwID : 1	
	LPSTR lpszName : "Name-1"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:12.046	
Out時刻	2002/03/25 22:24:12.057	
モジュール名	GetInfo.dll	3304
Interface名	GetInfo	
Method名	FreeNameBuffer	
In引数	LPSTR lpszName : "Name-1"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:12.068	
Out時刻	2002/03/25 22:24:12.079	
モジュール名	GetInfo.dll	3305
Interface名	GetInfo	
Method名	GetName	
In引数	DWORD dwID : 2	
	LPSTR lpszName : "Name-2"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:12.089	
Out時刻	2002/03/25 22:24:13.000	
モジュール名	GetInfo.dll	3306
Interface名	GetInfo	
Method名	FreeNameBuffer	
In引数	LPSTR lpszName : "Name-2"	
戻り値	HRESULT : 0x00000000(0)	
In時刻	2002/03/25 22:24:13.011	
Out時刻	2002/03/25 22:24:13.022	

【図 3 4】

```

[ uuid(58DB5633-0694-4340-97CE-4E1AC6BFFBA7), //TestDllStd.
  helpstring("TestDllStd Type Library For PAT"),
  version(1.0)
]

library TestDllStd
{
  typedef [public] struct
  {
    char chParam;
    unsigned char uchParam;
    short sParam;
    unsigned short usParam;
    int nParam;
    unsigned int unParam;
    long lParam;
    unsigned long ulParam;
    double dbParam;
    float fParam;
  } TESTSTRUCT;
  typedef [public] TESTSTRUCT *LPTESTSTRUCT;

//DEFINE_GUID(GUID_PROGID, 0x8e037d65, 0xefa0, 0x40e7, 0x91, 0x43, 0xef, 0x70, 0x56, 0x94, 0
0x79);
[ uuid(8E037D65-EFA0-40e7-9143-EF7056945B79),
  helpstring("TestDllStd.dll for PAT object." ),
]

  interface
    test
    {
      char _stdcall FuncCharStd([in] char chParam);
      char * _stdcall FuncPCharStd([in, out] char * lpchParam);

      TESTSTRUCT _stdcall FuncStructStd([in]TESTSTRUCT TestStruct);
      LPTESTSTRUCT _stdcall FuncPStructStd([in, out]LPTESTSTRUCT lpTestStruct);
    };
}

```

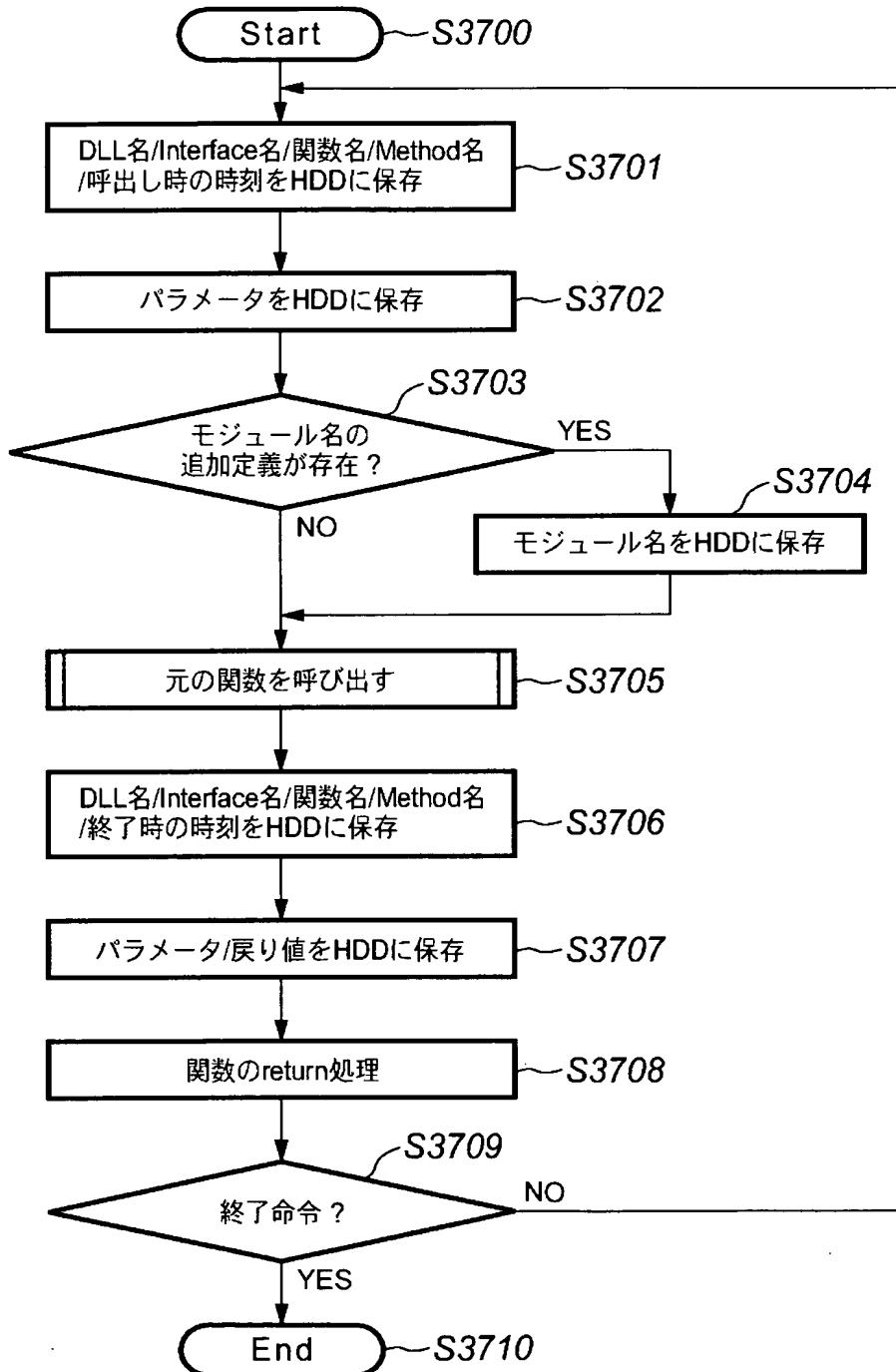
【図 3 5】

[Library] 3500  
 LibraryName=TestDllStd 3501  
 Patch=C: ¥Windows ¥System32 ¥spool ¥drivers ¥w32x86 ¥3  
 ModuleName=UnidrvUIPlugin.dll 3502  
 ...

【図36】

```
[Interface] 3600
InterfaceName=test 3601
Patch=C:\Windows\System32\spool\drivers\w32x86\3
ModuleName=UnidrvUIPlugin.dll
... 3602
```

【図37】



【書類名】 要約書

【要約】

【課題】 ソフトウェアの処理ログを容易に取得でき、かつ、バグの解析のための工数を削減することが可能なログ取得方法を提供する。

【解決手段】 関数を備えるプログラムのログ取得方法であって、プログラム実行時に呼び出されるOS内の関数のうち、指定された関数を識別する工程と、コードされた前記所定の処理を行う関数のアドレスと前記指定されたOS内の関数のアドレスとを、ログ取得のための関数のアドレスに書き換える工程とを備え、前記ログ取得のための関数は、前記所定の処理を行う関数と前記指定されたOS内の関数とを呼び出し、実行させた結果を前記プログラムに渡す工程（ステップS1708、S1712、S1713）と、前記所定の処理を行う関数と前記指定されたOS内の関数とを呼び出す際および結果を受け取った際の所定の情報を記録する工程（ステップS1706、S1709）とを備える。

【選択図】 図17

特願 2003-099465

出願人履歴情報

識別番号 [000001007]

1. 変更年月日 1990年 8月30日

[変更理由] 新規登録

住所 東京都大田区下丸子3丁目30番2号  
氏名 キヤノン株式会社